# Visual Studio.Net -C#
# 6. HAFTA
# System İsim Alanı , Temel Tür Yapıları, Kalıtım

# System İsim Alanı

- .NET sınıf kütüphanesinde yer alan System isim alanı içersinde oldukça kullanışlı bazı sınıflar bulunmaktadır.
- Bunlardan System.Array, System.Random, System.Convert, System.Math ve System.GC ’den daha önce bahsedilmişti.
- System isim alanı içerisinde yer alan temel veri tiplerinin sahip olduğu bazı metotlar da zaman zaman kullanılmaktadır.

# System İsim Alanı

- Tip.Parse(); : string biçimindeki verileri tip türüne çevirir.
- Nesne.CompareTo(object o); : metodu çağrılan nesne ile o nesnesini karşılaştırır. D e ğerler eşit ise 0, nesne değeri küçük ise negatif, büyük ise pozitif değer döndürür.
- Nesne.Equals(object o); : metodu çağrılan nesne ile o nesnesini karşılaştırır. Eşit ise true aksi halde false döndürür.
- Nesne.ToString(); : nesne değerini string şeklinde geri döndürür.

- int a=2; int b=32;
- b.CompareTo(a)
- b<a ise -1, b>a ise 1, b==a ise 0

- b. Equals(a)
- b==a ise true, b!=a ise false

# System İsim Alanı

- **Double** ve **float** değerler için bir önceki anlatılan metotlara ek olarak aşağıdaki metotlar da bulunur. Bu metotlar **true/false** yani boolean geri dönüş değerine sahiptir:
  - IsInfinity: Sayının sonsuzu temsil edip etmediğini kontrol eder.
  - IsNaN: Anlamlı bir sayı olup olmadığını kontrol eder.
  - IsPositiveInfinitiy, IsNegativeInfinity: Sayının + ya da - sonsuz olup olmadığını kontrol eder.
  - double.IsInfinity( double d);

- char veriler içinde bazı metotlar bulunur:
  - char.GetNumericValue(char x) : Eğer değer sayısal karakter içeriyorsa sayısal değeri aksi halde -1 döndürür.
  - Bunun dışında kullanılan metotlar tabloda veriliyor.

# System İsim Alanı

- Diğer metotlar,

| Metot | Koşul |
|---|---|
| **IsControl** | Karakter kontrol karakteri ise |
| **IsDigit** | Karakter bir rakam ise |
| **IsLetter** | Karakter bir harf ise |
| **IsLetterOrDigit** | Karakter bir harf ya da rakam ise |
| **IsLower** | Karakter küçük harf ise |
| **IsNumber** | Karakter rakam ise |
| **IsPunctuation** | Karakter noktalama işareti ise |
| **IsSeperator** | Karakter boşluk gibi ayırıcı ise |
| **IsSurroGate** | Karakter Unicode yedek karakteri ise |
| **IsSymbol** | Karakter sembol ise |
| **IsUpper** | Karakter büyük harf ise |
| **IsWhiteSpace** | Karakter tab ya da boşluk karakteri ise |

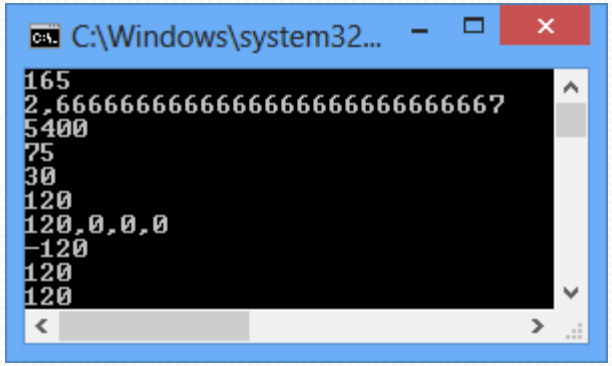
```
char.IsPunctuation("23.",2)→ True
char.IsPunctuation('2')→ False
```

# System İsim Alanı

- Decimal değerler içinde bazı metotlar mevcuttur:
- `Decimal.Add(d1, d2); //d1 ve d2'nin toplamını decimal türünden tutar. (+)`
- `Decimal.Divide(d1, d2); //d1'in d2'ye bölümünü decimal türünden tutar. (/)`
- `Decimal.Multiply(d1, d2); //d1 ile d2'nin çarpımını decimal türünden tutar. (*)`
- `Decimal.Subtract(d1, d2); //d1-d2'nin sonucunu decimal türünden tutar. (çıkarma işlemi) (-)`
- `Decimal.Remainder(d1, d2); //d1'in d2'ye bölümünden kalanı decimal türünden tutar. (mod alma işlemi)`
- `Decimal.Floor(d1); //d1'den büyük olmayan en büyük tam sayıyı decimal türünden tutar. (aşağı yuvarlama)`
- `Decimal.GetBits(d1); /*d1 için decimal sayı tanımlarken kullandığımız yapıcı işlevdeki beş parametreyi int türündeki bir dizi olarak tutar.*/`
- `Decimal.Negate(d1); //d1'in negatifini tutar.`
- `Decimal.Round(d1, sayi); /*sayi int türünde olmalıdır. Bu metot ile d1'in ondalık kısmındaki hane sayısı sayi kadar kalır. Yani d1 12.53666 ve sayi 3 ise 12.537 tutulur. Son kaybedilen hane 5 ya da 5'ten büyükse son kalan hane 1 artılır. sayi 0 olursa d1 tam sayıya yuvarlanmış olur.*/`
- `Decimal.Truncate(d1); //d1'in tam sayı kısmını tutar. Herhangi bir yuvarlama yapılmaz.`

# System İsim Alanı

```
int sayi = 5;  decimal d1 = 120, d2 = 45;
int[] dizi2 = Decimal.GetBits(d1);
Console.WriteLine(Decimal.Add(d1, d2) + "\n" +
Decimal.Divide(d1, d2) + "\n" +
Decimal.Multiply(d1, d2) + "\n" +
Decimal.Subtract(d1, d2) + "\n" +
Decimal.Remainder(d1, d2) + "\n" +
Decimal.Floor(d1) + "\n" +
dizi2[0] + "," + dizi2[1] + "," +
dizi2[2] + "," + dizi2[3]+ "\n" +
Decimal.Negate(d1) + "\n" +
Decimal.Round(d1, sayi) + "\n" +
Decimal.Truncate(d1));
```

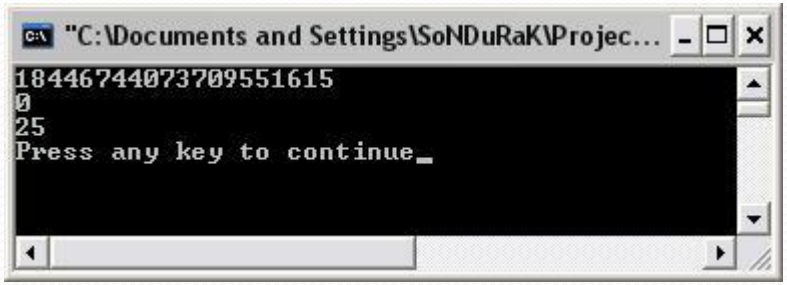


# C# Temel Tür Yapıları

- **Tamsayı Tipindeki Yapılar**

| Byte | sByte | İnt16 | uİnt16 |
|---|---|---|---|
| İnt32 | Uint32 | İnt64 | Uint64 |

```
using System;
class class1
{
    public static void Main()
    {
        string a = "25";
        int b = Int32.Parse(a);
        Console.WriteLine(UInt64.MaxValue);
        Console.WriteLine(UInt64.MinValue);
        Console.WriteLine(b);
    }
}
```



# C# Temel Tür Yapıları

- **Char Tipinde Yapılar**

| | |
|---|---|
| **ToLower(char str)** | str büyük harf ise küçük harfe çevirir. |
| **ToUpper(char str)** | str küçük harf ise büyük harfe çevirir. |

- `char aaa = 'a'; Console.WriteLine(char.ToUpper(aaa) + "\n");`
- Çıktı: A
- Diğer yapılar notlarda

- **Boolean Tipinde Yapılar**
- Boolean veri yapısında iki tane özellik vardır. Bunlar FalseString ve TrueString'dir. Bu özelliklerde String türündendir, false ve true yazılarını içerirler. CompereTo(), Equals(), Parse() ve ToString() boolean yapısında bulunan diğer metotlardır. Kendisine has bir metodu yoktur.

# C# Temel Tür Yapıları

- **DateTime ve TimeSpan**

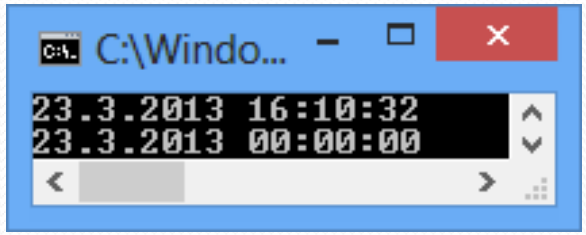
- C# dilinde tarih ve saat işlemleri System isim alanında bulunan DateTime ve TimeSpan yapıları ile gerçekleştirilir.

- DateTime yıl, ay, gün, saat, dakika, saniye gibi bilgileri tutan bir yapıdır.

- TimeSpan ise iki zaman bilgisi arasındaki farkı temsil etmek için kullanılır.

# DateTime

```
using System;
class zaman
{
    public static void Main()
    {
        DateTime tarihsaat = new DateTime();
        tarihsaat = DateTime.Now;
        Console.WriteLine(tarihsaat);
        DateTime tarih = new DateTime();
        tarih = DateTime.Today;
        Console.WriteLine(tarih);
    }
}
```

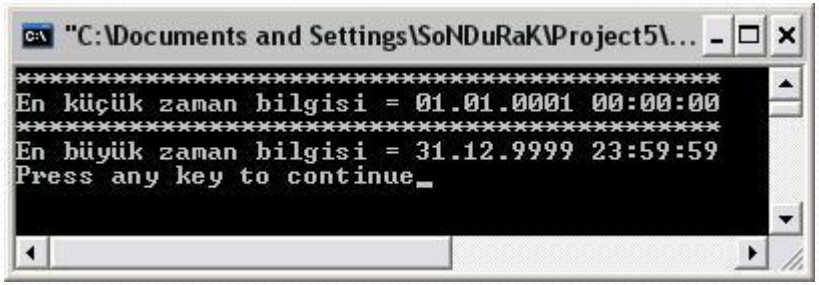


# DateTime

```
using System;

class Program
{
    static void Main()
    {
        Console.WriteLine(DateTime.MinValue);
        Console.WriteLine(DateTime.MaxValue);
        Console.WriteLine(DateTime.Now);
        Console.WriteLine(DateTime.Today);

        DateTime Bugun = new DateTime();

        Bugun = DateTime.Now;

        Console.WriteLine(Bugun.Date);
        Console.WriteLine(Bugun.Day);
        Console.WriteLine(Bugun.Month);
        Console.WriteLine(Bugun.Year);
        Console.WriteLine(Bugun.DayOfYear);
        Console.WriteLine(Bugun.DayOfWeek);
        Console.WriteLine(Bugun.TimeOfDay);
        Console.WriteLine(Bugun.Hour);
        Console.WriteLine(Bugun.Minute);
        Console.WriteLine(Bugun.Second);
        Console.WriteLine(Bugun.Millisecond);
        Console.WriteLine(Bugun.Ticks);
    }
}
```

# DateTime

```
using System;
class zaman
{
    public static void Main()
    {
     Console.WriteLine("*****************************");
     Console.WriteLine("En küçük zaman bilgisi = " + DateTime.MinValue);
     Console.WriteLine("*****************************");
     Console.WriteLine("En büyük zaman bilgisi = " + DateTime.MaxValue);
    }
}
```



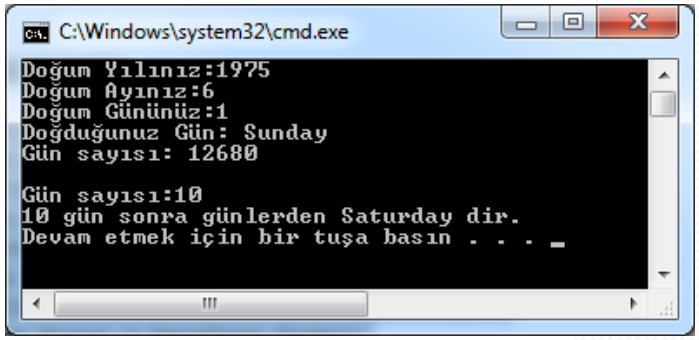
# DateTime ve TimeSpan

- **TimeSpan** ve **DateTime** yapıları ile tanımlanmış bazı operatörler bulunur.

- İki tarih arasındaki farkı bulmak için çıkarma (**-**), ileriki bir tarihi hesaplamak için toplama (**+**), iki tarih arasında büyüklük küçüklük karşılaştırması yapmak için de “**<**” ve “**>**” operatörleri aşırı yüklenmiştir.

# DateTime ve TimeSpan

```
using System;

class Program
{
    static void Main()
    {
        int yil, ay, gun;
        Console.Write("Doğum Yılınız:");
        yil = Convert.ToInt32(Console.ReadLine());
        Console.Write("Doğum Ayınız:");
        ay = Convert.ToInt32(Console.ReadLine());
        Console.Write("Doğum Gününüz:");
        gun = Convert.ToInt32(Console.ReadLine());

        DateTime Bugun = DateTime.Today;
        DateTime DogumGunu = new DateTime(yil, ay, gun);

        TimeSpan fark = Bugun - DogumGunu;

        Console.WriteLine("Doğduğunuz Gün: {0}", DogumGunu.DayOfWeek);
        Console.WriteLine("Gün sayısı: {0}", fark.Days);

        Console.WriteLine();
        Console.Write("Gün sayısı:");
        gun = Convert.ToInt32(Console.ReadLine());

        TimeSpan GunSayisi = new TimeSpan(gun, 0, 0, 0);
        DateTime Gelecek = DateTime.Today + GunSayisi;
        Console.WriteLine("{0} gün sonra günlerden {1} dir.", gun, Gelecek.DayOfWeek);
    }
}
```



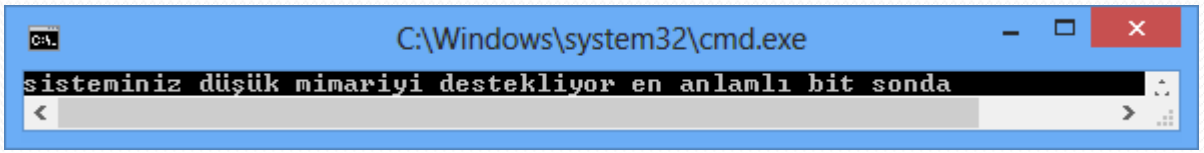
# System.BitConverter

- Alt seviye programlama da veriler byte dizisi şeklinde yani bitsel olarak işlenir.
- .NET içersinde de bite çevirmek amacıyla BitConverter sınıfı bulunur.
- Bu sınıfın IsLittleEndian isimli bir özelliği bulunur. Bu özellik işlemci mimarisine göre verileri bellekte depolarken hangi sıralamada yerleştirildiğini kontrol eder.
- En önemli metodu da GetBytes'tır. Amacı da farklı sayı türlerini byte dizisine çevirmektir.

# System.BitConverter

- Mimarinizi öğrenmek için;

```
using System;
class bitconverter
{
    public static void Main()
    {
        if (BitConverter.IsLittleEndian)
            Console.WriteLine("sisteminiz düşük mimariyi destekliyor en anlamlı bit sonda");
        else
            Console.WriteLine("sisteminiz yüksek mimariyi destekliyor en anlamlı bit başta");
    }
}
```



# System.BitConverter

- `int a=258; // 258/256=1 => 258-256^1 *1 = 2`
- `//00000000 00000000 00000001 00000010`
- $256^3$ $256^2$ $256^1$ $256^0$
- `byte[] dizi=BitConverter.GetBytes(a);`
- `foreach(byte b in dizi)`
- `{ Console.WriteLine(b); }`
- 2
- 1
- 0
- 0
- `byte[] dizi={2,1,0,0};` → en anlamlı bit en sonda
  `Console.WriteLine(BitConverter.ToInt32(dizi,0));`
- 258
- (256 ya bölünmeyene kadar sayıyı bölmeye devam et)

# System.Buffer

- Buffer sınıfı ile tür bilgisinden bağımsız bir biçimde byte düzeyinde veri işleme yapılır.

- Dizilerin belirli alanları başka bir diziye tür bilgisine bakılmaksızın aktarılabilir. Tüm veriler byte dizisi şeklinde düşünülür.

- BlockCopy, GetByte, SetByte en önemli metotlarıdır.

# System.Buffer

- `byte[] kaynak={1,2,3,1};`
- `//00000001 , 00000010 , 00000011 , 00000001`
- `short[] hedef=new short[4];`
- `//0000000000000000 , 0000000000000000 , 0000000000000000 , 0000000000000000`
- `Buffer.BlockCopy(kaynak,0,hedef,0,4);`
- `/*hedef dizisinin yeni hâli:`
- `0000001000000001 , 0000000100000011 , 0000000000000000 , 0000000000000000*/`
- `foreach(short a in hedef)  Console.Write(" "+a);`
- Bu program, mimarimiz Little Endian ise ekrana: 513 259 0 0 yazar.
- Burada derleyici her elemanı bellekte 1 bayt yer kaplayan kaynak dizisinin elemanlarını her elemanı bellekte 2 bayt kaplayan hedef dizisine olduğu gibi kopyaladı. Derleyici burada karşılaştığı ilk baytı düşük anlamlı bayta, ikinci baytı da yüksek anlamlı bayta kopyaladı.

# System.Buffer

- **ByteLength() metodu**
- Kendisine parametre olarak verilen bir dizideki toplam bayt sayısını bulur. Örneğin kendisine parametre olarak gönderilen 3 elemanlı short türünden bir dizi sonucu 6 sayısı gönderir. Geri dönüş tipi inttir. Örnek:
  - `short[] dizi=new short[4];`
  - `Console.WriteLine(Buffer.ByteLength(dizi)); // 8 yazar`
- **GetByte() metodu**
  - `static byte GetByte(Array dizi,int a)`
  - dizi dizisinin a. baytını verir.
- **SetByte() metodu**
  - `static void SetByte(Array dizi,int a,byte deger)`
  - a. baytı deger olarak değiştirir.

# System.Buffer

- Örnek:
  - byte[] dizi={0,3,2,1,4}; Console.WriteLine(Buffer.GetByte(dizi,3));
  - Bu kod 1 değerini döndürür. Eğer dizinin tipi byte değil de short olsaydı işler değişirdi. Çünkü o zaman hem sıfırla beslenen baytlar da hesaba katılırdı ve hem de bilgisayarımızın mimarisi sonucu etkilerdi. Bu durumu örneklendirelim:
  - short[] dizi={0,3,2,1,4}; *// 0-> 0000000000000000 ,2->0000000000000011*
  - *4->0000000000000010 ,6->0000000000000001, 8->0000000000000100*
    Console.WriteLine(Buffer.GetByte(dizi,4));
  - Bu kod Little Endian mimarisinde ekrana 2 yazar. Mimari Big Endian olsaydı ekrana 0 yazacaktı. Çünkü Little Endian mimarisinde verilerin önce düşük anlamlı baytı okunur/yazılır. Big Endian mimarisinde ise tam tersi söz konusudur.
  - Console.WriteLine(Buffer.GetByte(dizi,8));// Ekrana 4 yazar
- Örnek:
  - byte[] dizi={2,1,4}; Buffer.SetByte(dizi,1,5); Console.WriteLine(dizi[1]); *//Ekrana 5 yazılır.*
  - Yine eğer dizinin türü byte değil de int olsaydı işler değişirdi.

# C# Temel Tür Yapıları

- **Convert Sınıfı**

```
using System;
class ortalamaHesapla
{   public static void Main()
    {   Console.Write("Vize1 = ");
        int vize1 = Convert.ToInt32(Console.ReadLine());
        Console.Write("Vize2 = ");
        int vize2 = Convert.ToInt32(Console.ReadLine());
        Console.Write("Final = ");
        int final = Convert.ToInt32(Console.ReadLine());
        double a = ((double)vize1 * 0.2 + (double)vize2 * 0.3 +
(double)final * 0.5);
        Console.WriteLine("Ortalama = {0}", a);
    }
}
```

# String İşlemleri

- Programlarda kullanılan verilerin büyük bir çoğunluğu string türündendir.
- C# da string işlemleri System.String sınıfı içersinde yer alan üye metot ve özelliklerle yapılır.
- String veriler için indeksleyici de tanımlanmıştır. Yani bir karakter dizisi gibi işlem görebilir. Fakat karakterler salt okunurdur.
- Stringler değişik şekillerde tanımlanabilirler:
- `string a = “C# Programlama Dili”;`
- `char[] dizi = {’1’,’2’,’3’,’4’,’5’};`
- `String s = new String(dizi);//12345`
- `String s = new String(dizi,1,2); //dizi[1]'den itibaren 2 eleman stringe atandı. → 23`
- `String s = new String(‘x’,10);//xxxxxxxxxx`

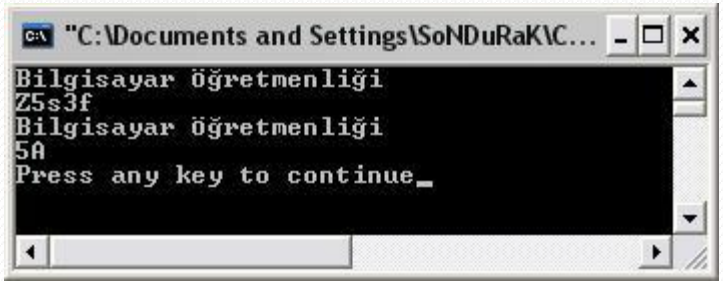
# String İşlemleri

- String sınıfının metotları oldukça fazladır. Önemli metotlarından bazıları şunlardır:
- string.Concat()
  - `static string Concat(params Array stringler)`
  - String verilerin ardarda eklenmesini sağlar. + operatörü ile eşdeğerdir.
- string.Compare()
  - İki string değeri karşılaştırı. == ve != operatörleri ile benzer işlem gerçekleştirir. Fakat Compare metodunun bazı aşırı yüklemeleri ile daha gelişmiş (büyük küçük harf duyarlılığı gibi.) karşılaştırmalar yapılabilir.
  - **static int** Compare(**string** a,**string** b)
  - **static int** Compare(**string** a,**string** b,**bool** c)
  - **static int** Compare(**string** a,**int** indeks1,**string** b,**int** indeks2)
  - **static int** Compare(**string** a,**int** indeks1,**string** b,**int** indeks2,**bool** c)
  - kıyaslamada ilk elemanların a[indeks1] ve b[indeks2] sayılmasıdır. Yani stringlerdeki bu elemanlardan önceki elemanlar yok sayılır.

# String İşlemleri

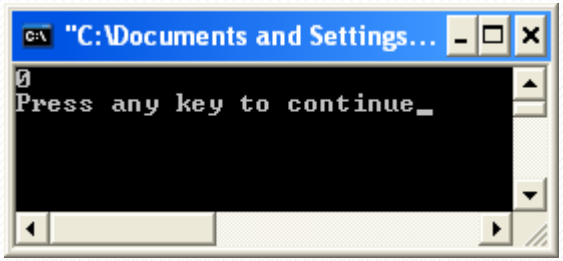
- String.Concat()

```
using System;
class Concat
{    public static void Main()
    {   String str1 = String.Concat("Bilgisayar", " Öğretmenliği");
        Console.WriteLine(str1);
        String str2 = String.Concat("Z", "5", "s", "3", "f");
        Console.WriteLine(str2);
        String str3 = "Bilgisayar" + " Öğretmenliği";
        Console.WriteLine(str3);
        String str4 = String.Concat(5, "A");
        Console.WriteLine(str4);
    }
}
```



# String İşlemleri

- String.Compare()
- using System;
- class class1 {
- public static void Main() {
- string a="Aa";
- bool sa=true;//false büyük küçük duyarsız
- string v="aa";
- int c=String.Compare(a,v,sa);
- Console.WriteLine(c);
- 
- }
- }



# String İşlemleri

- stringnesne.CompareTo(string str)
- stringnesne.IndexOf()
  - string içersinde alt stringlerin aranmasını sağlar. Geriye aranan alt stringin bulunduğu konumu ya da bulunamadı (-1) bilgisini döndürür.
  - int IndexOf(string a)
  - int IndexOf(char b)
- stringnesne.LastIndexOf()
  - IndexOf ile aynı çalışır Farkı ise aranan karakterin en son nerede görüldüğünün indeksini geri döndürür.
  - int LastIndexOf(string a)
  - int LastIndexOf(char b)

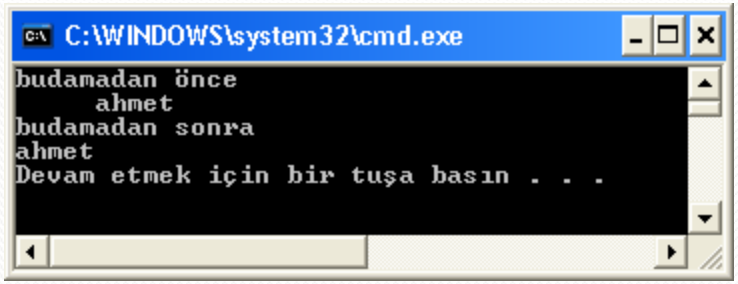
# String İşlemleri

- IndexOf
- using System;
- class arama{
- public static void Main() {
- string yazı="firat üniversitesi";
- Console.WriteLine(yazı.IndexOf("ver"));
- Console.WriteLine(yazı.IndexOf('t'));
- Console.WriteLine(yazı.IndexOf('c'));
- }
- }
- Diğer arama metotları ders notlarında mevcut

# String İşlemleri

- stringnesne.Trim()
  - Bir string ifadenin başındaki ve sonundaki boşlukları ya da belirlenmiş karakterleri atar.

```
using System;
class tarih  {
    public static void Main()
    {   string a = "     ahmet    ";
        Console.WriteLine("budamadan önce");
        Console.WriteLine(""+a);
        a = a.Trim ();
        Console.WriteLine("budamadan sonra");
        Console.Write("" + a);
    }
}
```

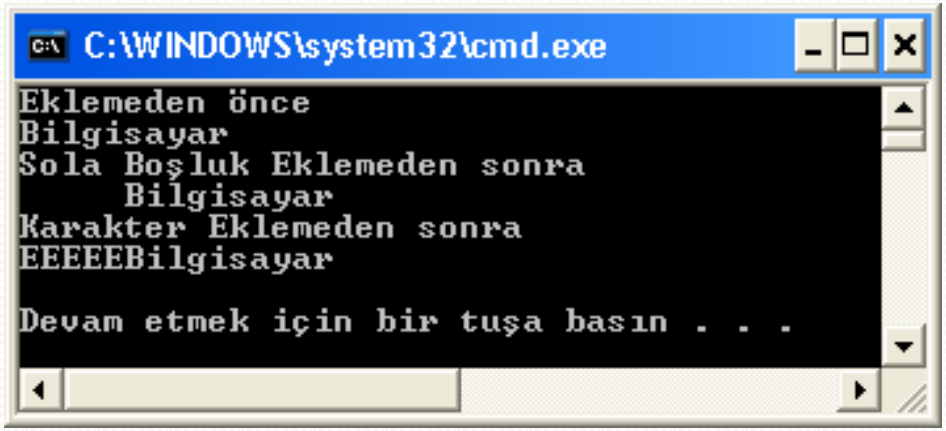


# String İşlemleri

- PadRight, PadLeft
  - Bir stringin sağına ya da soluna yeni karakterler ilave etmek için kullanılır.
  - string PadRight(int boyut);
    - stringin uzunluğu boyuta eşit olana kadar stringin sağına boşluk ekler
  - string PadRight(int boyut,char a);
    - stringin uzunluğu boyuta eşit olana kadar stringin sağına 'a' karakterini ekler
  - string PadLeft(int boyut);
    - stringin uzunluğu boyuta eşit olana kadar stringin soluna boşluk ekler
  - string PadLeft(int boyut,char a);
    - stringin uzunluğu boyuta eşit olana kadar stringin soluna 'a' karakterini ekler

# String İşlemleri

```
using System;
class tarih
{
    public static void Main()
    {
        string a = "Bilgisayar";

        Console.WriteLine("Eklemeden önce");
        Console.WriteLine(""+a);
        a = a.PadLeft(15);
        Console.WriteLine("Sola Boşluk Eklemeden sonra");
        Console.WriteLine("" + a);
        Console.WriteLine("Karakter Eklemeden sonra");
        a = "Bilgisayar";
        a = a.PadLeft(15,'E');
        Console.WriteLine("" + a);
        Console.WriteLine("");
    }
}
```

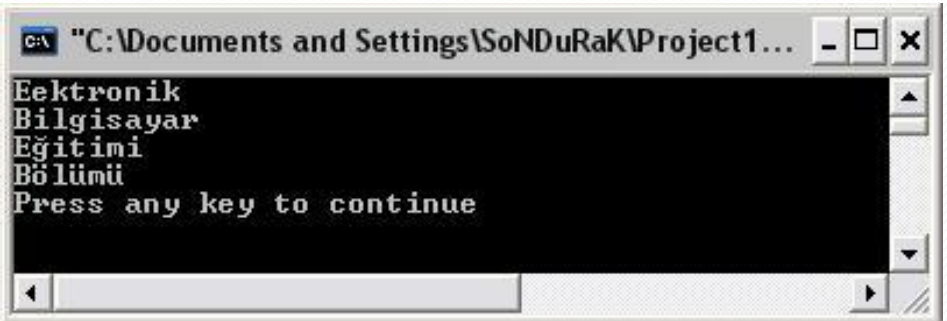


# String İşlemleri

- **bool string.StartsWith(string a)**
- Metodu çağıran "**string a**" ile başlıyorsa **true**, diğer durumlarda **false** değeri üretir.
- **bool string.EndsWith(string b)**
- Metodu çağıran "**string b**" ile bitiyorsa **true**, diğer durumlarda **false** değeri üretir.
- stringnesne.Split()
  - Belli bir biçime sahip olan toplu string verileri, belirtilen ayırıcı karakterlerden ayırıp ayrı bir string dizisi üretir.
    - **string [] Split(params char[] ayırıcı)**
    - **string [] Split(params char[] ayırıcı, int toplam)**
    - İkinci metotta ise bu parçalama işlemi en fazla toplam sayısı kadar yapılır.
- string.Join()
  - Split metodunun tersi gibi çalışır. Ayrı stringleri belli bir ayırıcı karakter ile birleştirip tek bir string üretir.
    - **static string join(string ayırıcı,string[] yazılar)**
    - **static string join(string ayırıcı,string[] yazı,int başla,int toplam)**
    - İkinci metotta ise [basla] ‘dan itibaren toplam kadar yazı elemanı birleştirilir.

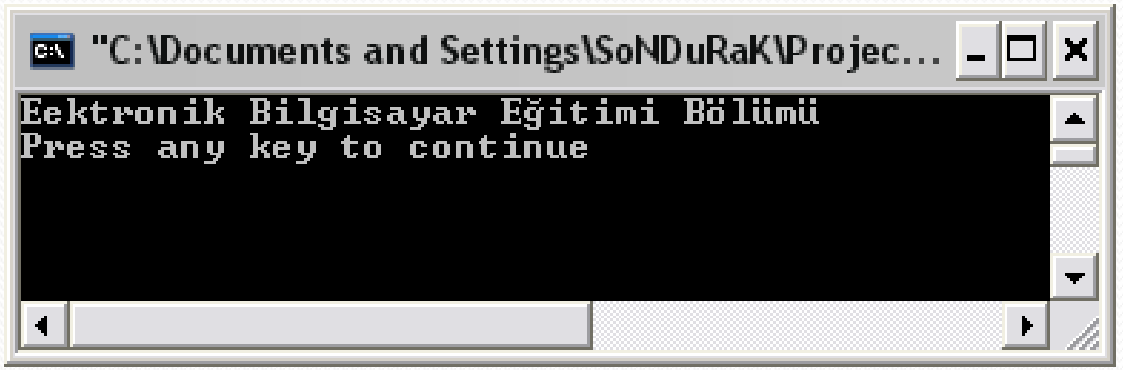
# String İşlemleri

```
using System;
class Ayırmaişlemi
{
        public static void Main()
        {
                string str="Eektronik Bilgisayar Eğitimi Bölümü";
                char[] ayırıcı={' '};
                string[] ayır=str.Split(ayırıcı);
                foreach(string i in ayır)
                        Console.WriteLine(i);
        }
}
```



# String İşlemleri

```
using System;
class bireştirmeişlemi
{
        public static void Main()
        {
                string[] str={"Eektronik" ,"Bilgisayar" ,"Eğitimi" ,"Bölümü"};
                string birleş=String.Join(" ",str);
                Console.WriteLine(birleş);
        }
}
```

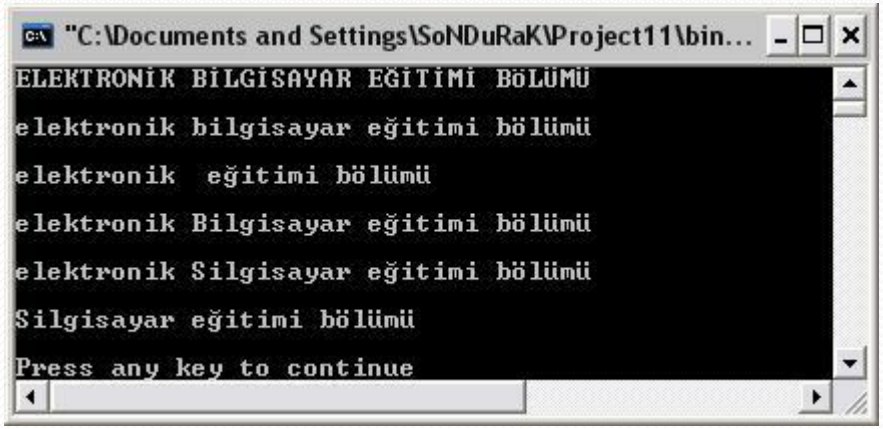


# String İşlemleri

- stringnesne.ToUpper()
  - Stringin karakterlerinin hepsini büyük harfe çevirip geri döndürür.
- stringnesne.ToLower()
  - Stringin karakterlerinin hepsini küçük harfe çevirip geri döndürür.
- stringnesne.Remove(**int** indeks, **int** adet)
  - String içinden belli sayıda karakterin atılmasını sağlar.
- stringnesne.Insert(**int** indeks,**string** str)
  - String içine belirli indeksten itibaren yeni bir string eklemek için kullanılır.
- stringnesne.Replace(**char |string** c1,**char|string** c2)
  - String içindeki belirlenen karakter ya da karakterleri başkalarıyla değiştirir.
- stringnesne.SubString(**int** indeks |**int** indeks,**int** toplam)
  - String ifadenin belli bir kısmının elde edilmesini sağlar.

# String İşlemleri

```
using System;
class digermetotlar
{
        public static void Main()
        {
        string str="Elektronik Bilgisayar Eğitimi Bölümü";
        str=str.ToUpper();
        Console.WriteLine(str+"\n");
        str=str.ToLower();
        Console.WriteLine(str+"\n");
        str=str.Remove(11,10);
        Console.WriteLine(str+"\n");
        str=str.Insert(11,"Bilgisayar");
        Console.WriteLine(str+"\n");
        str=str.Replace('B','S');
        Console.WriteLine(str+"\n");
        str=str.Substring(11);
        Console.WriteLine(str+"\n");//11. karakterlerden sonrasını göster
        }
}
```



# Biçimlendirme

- Program çıktılarının belli bir düzende olması oldukça önemlidir. Bazı zamanlar standart çıktıların anlaşılması zor olabilir.

- Bu problemi çözmek için biçimlendirme teknikleri kullanılır. Bu teknikler yalnızca biçimlendirmeyi destekleyen komutlar tarafından kullanılabilir.

- **Console.WriteLine(), String.Format()** ve **ToString** metotları biçimlendirmeyi destekleyen metotlardır.

- Bu metotlarda kullanılan **{ }** parantezleri arasındaki ifadeler belli değişkenlerin belli bir düzende biçim metnine aktarılmasını sağlıyordu:
  - Console.WriteLine(“Merhaba **{**0**}**!”,isim)

# Biçimlendirme

- En genel kullanımı
  - **{ değişken_no, genişlik : format}**
  - **int** a=54;
- Sadece değişken verildiğinde değişkenin türüne göre varsayılan ayarlar kullanılır
- Genişlik yazılacak olan verinin diğer verilerle olan mesafesini ve hangi yöne hizalanması gerektiğini belirler. Format ise verinin türüne göre değişik biçimlendirilmesini sağlar.
  - Console.WriteLine("{0,10} numara",a);
  - Console.WriteLine("{0,-10} numara",a);
  - 54 numara
  - 54 numara

# Biçimlendirme

| Belirleyici | Açıklama | Duyarlılık Anlamı |
|---|---|---|
| C/c | Para birimi | Ondalık basamakların sayısını verir. |
| D/d | Tam sayı verisi | En az basamak sayısını belirtir, gereğinde bos olan basamaklar sıfır ile beslenir. |
| E/e | Bilimsel notasyon | Ondalık basamak sayısını verir. |
| F/f | Gerçek sayılar (float) | Ondalık basamak sayısını verir. |
| G/g | E ve F biçimlerinden hangisi kısa ise o kullanılır | Ondalık basamak sayısını verir. |
| N/n | Virgül kullanarak gerçek sayıları yazar | Ondalık basamak sayısını verir. |
| P/p | Yüzde | Ondalık basamak sayısını verir. |
| X/x | Onaltılık sayı sisteminde yazar. | En az basamak sayısını belirtir, gereğinde boş olan basamaklar sıfırla beslenir. |

# Biçimlendirme

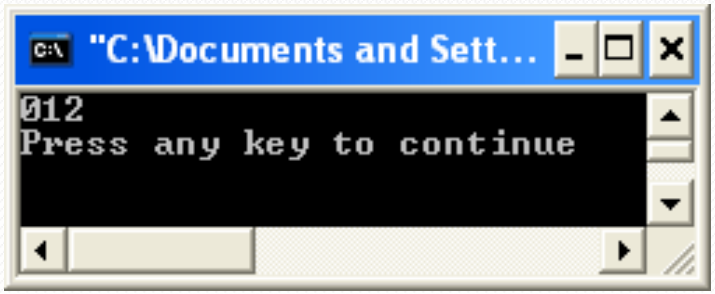
```
using System;
class Formatlama {
    static void Main() {
        float f=568.87f;
        int a=105;
        Console.WriteLine("{0:C3}",a);
        Console.WriteLine("{0:D5}",a);
        Console.WriteLine("{0:E3}",f);
        Console.WriteLine("{0:F4}",f);
        Console.WriteLine("{0:G5}",a);
        Console.WriteLine("{0:N1}",f);
        Console.WriteLine("{0:P}",a);
        Console.WriteLine("{0:X5}",a);
        Console.WriteLine("{0:C3}",f); } }
```

```
105,000 TL
00105
5,689E+002
568,8700
105
568,9
%10.500,00
00069
568,870 TL
```

# Biçimlendirme

- String.Format() ve ToString() Metotları İle Biçimlendirme

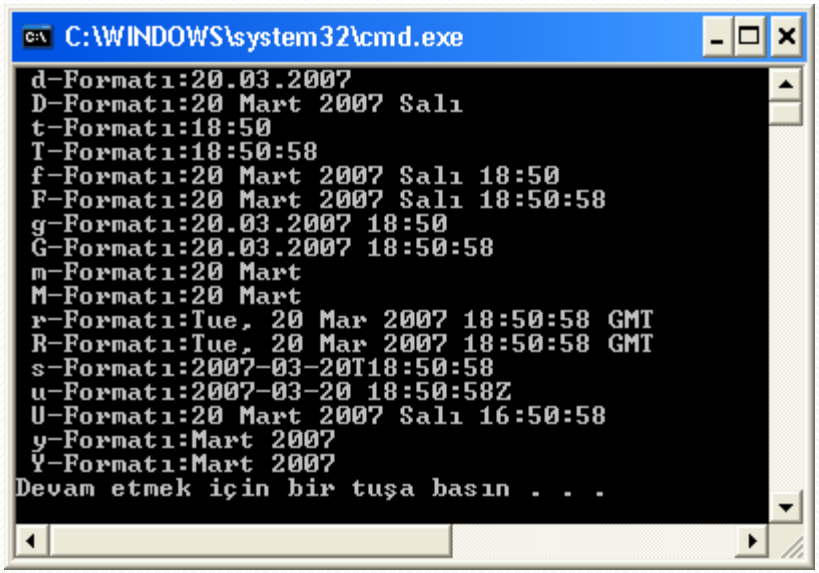
```
using System;
class StringFormat
{       public static void Main()
        {       int a=12;
                string str=String.Format("{0:d3}",a);
                Console.WriteLine(str);
        }
}
```




```
using System;
class Tostring
{       public static void Main()
        {       int a=12;
                string str=a.ToString("C3");// (para biçimi)
                Console.WriteLine(str);
        }
}
```

# Tarih ve Saat Biçimlendirme

```
using System;
class TarihveSaat {
  public static void Main()                {
            DateTime d=DateTime.Now;
            Console.WriteLine(" d-Formatı:{0:d}",d);
            Console.WriteLine(" D-Formatı:{0:D}",d);
            Console.WriteLine(" t-Formatı:{0:t}",d);
            Console.WriteLine(" T-Formatı:{0:T}",d);
            Console.WriteLine(" f-Formatı:{0:f}",d);
            Console.WriteLine(" F-Formatı:{0:F}",d);
            Console.WriteLine(" g-Formatı:{0:g}",d);
            Console.WriteLine(" G-Formatı:{0:G}",d);
            Console.WriteLine(" m-Formatı:{0:m}",d);
            Console.WriteLine(" M-Formatı:{0:M}",d);
            Console.WriteLine(" r-Formatı:{0:r}",d);
            Console.WriteLine(" R-Formatı:{0:R}",d);
            Console.WriteLine(" s-Formatı:{0:s}",d);
            Console.WriteLine(" u-Formatı:{0:u}",d);
            Console.WriteLine(" U-Formatı:{0:U}",d);
            Console.WriteLine(" y-Formatı:{0:y}",d);
            Console.WriteLine(" Y-Formatı:{0:Y}",d); }
}
```



# Biçimlendirme

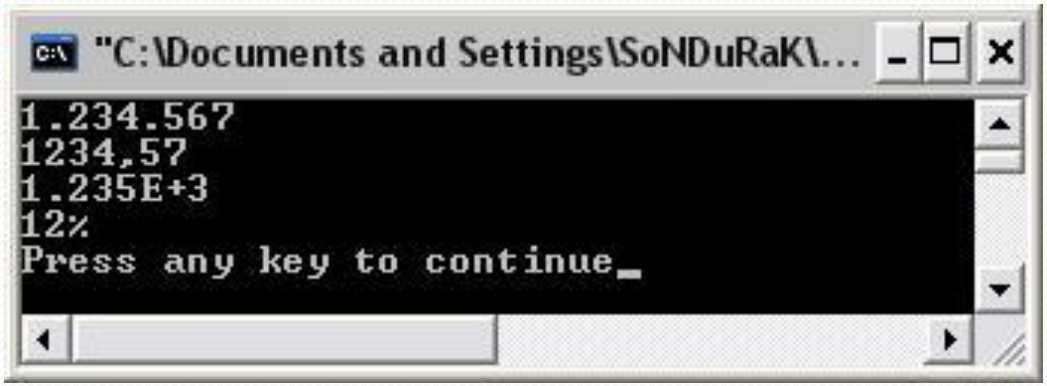
- Standart biçimlendirmelerin dışında özel biçimlendirmeler de tasarlanabilir.
- **“#” “,” “.” “0” “E”** ve **“%”** karakterleri ile özel biçimlendirmeler oluşturulabilir.

  - **#** → Rakam değerleri için kullanılır.
  - **,** → Büyük sayılarda binlikleri ayırmak için kullanılır.
  - **.** → Gerçek sayılarda ondalıklı kısımları ayırmak için kullanılır.
  - **0** → Yazılacak karakterin başına ya da sonuna 0 ekler.
  - **%** → Yüzde ifadelerini belirtmek için kullanılır.
  - **E0**, **e0**, **E+0**, **e+0**, **E-0**, **e-0** → Sayıları bilimsel notasyonda yazmak için kullanılır.

  - **{0:#,###.##}**
  - **{0:#%}**
  - **{0:#,###E+0}**

# Biçimlendirme

```
using System;
 class özelbiçimlendirme
{
        public static void Main()
        {       Console.WriteLine("{0:#,###}",1234567);
                Console.WriteLine("{0:#.##}",1234.5678);
                Console.WriteLine("{0:#,###E+0}",1234567);
                Console.WriteLine("{0:#%}",0.123);
        }
}
```



# Biçimlendirme

- **Düzenli İfadeler**
- Düzenli ifadeler değişken sayıda karakterden oluşabilecek ancak belirli koşulları sağlayan ifadelerdir.

- Örneğin e-posta adreslerini düşünebiliriz. Dünyada milyonlarca e-posta adresi olmasına ve farklı sayıda karakterden oluşabilmesine rağmen hepsi kullaniciadi@domainismi.domain tipi düzenindedir.

- Örneğin iletisim@microsoft.com bu düzenli ifadeye uymaktadır.
- C#'taki düzenli ifade işlemleri temel olarak System.Text.RegularExpressions isim alanındaki Regex sınıfı ile yapılmaktadır.

- Bir karakter dizisinin oluşturulan düzenli ifadeye uyup uymadığını yine bu isim alanındaki Match sınıfıyla anlarız.

- Düzenli ifadeler başlı başına bir kitap olabilecek bir konudur. Burada sadece ana hatları üzerinde durulacaktır.

# Biçimlendirme

- **Düzenli İfadelerin Oluşturulması**
- Bir ifadenin mutlaka istediğimiz karakterle başlamasını istiyorsak ^ karakterini kullanırız. Örneğin ^9 düzenli ifadesinin anlamı yazının mutlaka 9 karakteri ile başlaması demektir. "9Abc" yazısı bu düzene uyarken "f9345" bu düzene uymaz.
- Belirli karakter gruplarını içermesi istenen düzenli ifadeler için \ karakteri kullanılır:
  - **\D** ifadesi ile yazının ilgili yerinde rakam olmayan tek bir karakterin bulunması gerektiği belirtilir.
  - **\d** ifadesi ile yazının ilgili yerinde 0-9 arası tek bir karakterin bulunması gerektiği belirtilir.
  - **\W** ifadesi ile yazının ilgili yerinde alfanumerik olmayan karakterin bulunması gerektiği belirtiliyor. Alfanumerik karakterler a-z, A-Z ve 0-9 aralıklarındaki karakterlerdir.
  - **\w** ifadesi ile yazının ilgili yerinde bir alfanumerik karakterin bulunması gerektiği belirtiliyor.
  - **\S** ifadesi ile yazının ilgili yerinde boşluk veya tab karakterleri haricinde bir karakterin olması gerektiği belirtiliyor.
  - **\s** ifadesi ile yazının ilgili yerinde yalnızca boşluk veya tab karakterlerinden biri bulunacağı belirtiliyor.

# Biçimlendirme

- Bu öğrendiğimiz bilgiler ışığında 5 ile başlayan, ikinci karakteri rakam olmayan, üçüncü karakteri ise boşluk olan bir düzenli ifade şöyle gösterilebilir:
  - ^5\D\s
  - Tahmin edersiniz ki aynı zamanda burada düzenli ifademizin yalnızca 3 harfli olabileceği belirttik. Yukarıdaki ^5\D\s ifadesine filtre denilmektedir.
- Belirtilen gruptaki karakterlerden bir ya da daha fazlasının olmasını istiyorsak + işaretini kullanırız.
  - \w+
  - filtresi ilgili yerde bir ya da daha fazla alfanumerik karakterin olabileceğini belirtiyor. "123a" bu filtreye uyarken "@asc" bu filtreye uymaz. + yerine * kullansaydık çarpıdan sonraki karakterlerin olup olmayacağı serbest bırakılırdı.
- Birden fazla karakter grubundan bir ya da birkaçının ilgili yerde olacağını belirtmek için | (mantıksal veya) karakteri kullanılır. Örnek:
  - m|n|s
  - ifadesinde ilgili yerde m, n ya da s karakterlerinden biri olmalıdır. Bu ifadeyi parantez içine alıp sonuna + koyarsak bu karakterlerden biri ya da birkaçının ilgili yerde olacağını belirtmiş oluruz:
  - (m|n|s)+

# Biçimlendirme

- Sabit sayıda karakterin olmasını istiyorsak {adet} şeklinde belirtiriz. Örnek:
  - \d{3}-\d{5}
  - filtresine "215-69857" uyarken "54-34567" uymaz.
- ? karakteri, hangi karakterin sonuna gelmişse o karakterden en az sıfır en fazla bir tane olacağı anlamına gelir. Örnek:
  - \d{3}B?A
  - Bu filtreye "548A" veya "875BA" uyarken "875BBA" uymaz.
- . (nokta) işareti ilgili yerde "\n" dışında bir karakterin bulunabileceğini belirtir. Örneğin
  - \d{3}.A
  - filtresine "123sA" ve "8766A" uyar. "236\nA"; uymaz
- \b bir kelimenin belirtilen yazıyla sonlanması gerektiğini belirtir. Örnek:
  - \d{3}dır\b
  - filtresine "123dır" uyarken "123dırb" uymaz.

# Biçimlendirme

- \B ile bir kelimenin başında ya da sonunda bulunmaması gereken karakterler belirtilir. Örnek:
- \d{3}dır\B
  - filtresine "123dır" veya "dır123" uymazken "123dır8" uyar.
- Köşeli parantezler kullanarak bir karakter aralığı belirtebiliriz. Örneğin ilgili yerde sadece büyük harflerin olmasını istiyorsak [A-Z] şeklinde, ilgili yerde sadece küçük harfler olmasını istiyorsak [a-z] şeklinde, ilgili yerde sadece rakamlar olmasını istiyorsak [0-9] şeklinde belirtebiliriz. Ayrıca sınırları istediğimiz şekilde değiştirebiliriz. Örneğin [R-Y] ile ilgili yerde yalnızca R ve Y arası büyük harfler olabileceğini belirtiriz.

- **Regex sınıfı**
- Regex sınıfı bir düzenli ifadeyi tutar. Bir Regex nesnesi şöyle oluşturulur:
- Regex nesne=new Regex(string filtre);
- Yani bu yapıcı metoda yukarıda oluşturduğumuz filtreleri parametre olarak veririz. Regex sınıfının Match metodu ise kendisine gönderilen bir yazının düzenli ifadeye uyup uymadığını kontrol eder ve uyan sonuçları Match sınıfı türünden bir nesne olarak tutar.

# Biçimlendirme

- **Match sınıfı**
- Match sınıfının NextMatch() metodu bir Match nesnesindeki bir sonraki düzenli ifadeyi döndürür. Yazının düzenli ifadeye uyup uymadığının kontrolü ise Match sınıfının Success özelliği ile yapılır.
- Eğer düzenli ifadeye uygun bir yapı varsa Success özelliğinin tuttuğu değer true olur.
- Örnek:
- A\d{3}(a|o)+
- Bu filtreyle düzenli ifademizin şöyle olduğunu söyleyebiliriz:
- İlk karakter A olacak.
- A'dan sonra üç tane rakam gelecek.
- Üç rakamdan sonra "a" ya da "o" karakterlerinden biri ya da birkaçı gelecek.

# Biçimlendirme

```
using System;
using System.Text.RegularExpressions;
class duzenli
{  static void Main()
   {  string filtre=@"A\d{3}(a|o)+";
      Console.Write("Yazı girin: ");
      string yazi=Console.ReadLine();
      Regex nesne=new Regex(filtre);
      Match a=nesne.Match(yazi);
      Console.WriteLine(a.Success);
      Console.WriteLine(a.ToString());
      Console.WriteLine(a.Index);
      Console.WriteLine(a.NextMatch());
     Console.WriteLine(a.Index);
   }
}
```

Bu programda kullanıcının

**A123aA657oA456oao**

girdiğini varsayarsak ekran çıktısı şöyle olur.

```
True
A123a
0
A657o
10
```

# Nesne Yönelimli Programlama ve Kalıtım

# Nesne Yönelimli Programlama

- Modern dillerin bir çoğunda nesneye yönelimli programlama tekniği desteklenmektedir. Bu teknik yazılım geliştirmeyi kısaltan ve sistematik hale getiren bir yapıdır

- C# dili de bu tekniği tamamıyla desteklemektedir. Nesne yönelim tekniği, gerçek hayatı programlar için simule edecek yöntemlerin birleşimidir.

- Bu teknikte geliştirilmek istenen sistem parçalara ayrılır ve bu parçalar arasında ilişkiler kurulur. Parçalar hiyerarşik ya da bağımsız olabilir.

- Bağımsız bileşenler birbirleriyle haberleşerek etkileşimde bulunurlar.

# Nesne Yönelimli Programlama

- Nesne yönelimli programlama kavramlarından bazıları şunlardır:

- Nesne yönelimli programlama tekniğinin en temel bileşeni nesnelerdir. Nesneler içeriklerinde veriler barındırırlar. Veriler arası ilişkiler sağlayan fonksiyonlara da sahiptirler. Nesnelerin veri ve fonksiyon gibi bileşenleri içermesine sarmalama (encapsulation) denilir.

- Nesne içindeki veriler ve fonksiyonlar nesnenin dışarıya nasıl hizmet verdiğini belirler. Fakat bu hizmeti nasıl verdiği belli değildir. Nesnenin hizmetlerinden faydalanmak için nesnenin dış dünyadan erişilen ara yüzünün bilinmesi yeterlidir. Buna bilgi saklama (information hiding) adı verilir.

- Nesnelerin birbirlerinden bağımsız olmasına rağmen aralarında haberleşebilirler. Hangi nesnenin hangi nesneye mesaj göndereceği, hangi nesnelerin fonksiyonlarının kullanılacağı derleme aşamasında belli olmayabilir. Bu durumda geç bağlama (late binding) mekanizmasından faydalanılır.

# Nesne Yönelimli Programlama

- Tüm nesneler birer sınıf örneğidir. Sınıflar nesnelerin özelliklerini belirlerler. Nesneler derleme ya da çalışma anında oluşturulabilir.

- Kalıtım ile nesneler birbirinden türetilebilir. Türeyen sınıf diğer sınıfın tüm özelliklerini ve kendine has özellikleri içerebilir. Kalıtım yolu ile türetilmiş sınıflar ile hiyerarşik sınıf organizasyonu gerçekleştirilebilir.

- Nesneye yönelimli programlama tekniğinde nesneler çok biçimli olabilir. Çok biçimlilik (polymorphism) kavramı türeme ile alakalıdır ve anlamı bir nesnenin farklı şekillerde davranabilmesidir.

# Nesne Yönelimli Programlama

- **Nesne Kavramı**

- Gerçek dünyadaki varlığını bildiğimiz bir çok şey birer nesnedir. Nesne yönelimli programlama tekniğinde de sınıflar nesnelerin biçimlerini belirlerler. Oluşturulan nesneler sınıf türünden nesne olarak adlandırılır. Her nesne kendi içinde tutarlı bir yapıya sahiptir yani veriler arasında sıkı bir bağ bulunur ki bu nesne mantığının temelidir.

- Sınıflardan nesneler oluşturmak için new anahtar sözcüğü kullanılıyordu.
  - Sinif nesne = new Sinif();

- Sınıflar nesnelerin şeklini belirlerler. Yani nesnenin türünü tanımlarlar. Kısaca sınıflar bir tür bilgisidir.

# Kalıtım (Inheritance)

- Kalıtım nesne yönelimli programlama tekniğinin en önemli özelliğidir. Kalıtım yolu ile sınıflar birbirinden türetilir.

- Türeyen sınıflar türedikleri sınıfın özelliklerini kalıtım yoluyla devralırlar ve kendisi de yeni özellikler tanımlayabilir.

- Türetme ile sınıflar arasında hiyerarşik bir yapı kurulabilir.

- Örnek vermek gerekirse dünya üzerinde yaşayan canlıları sınıflandırmak mümkün. Bu sınıflardan bir tanesi de hayvanlar olabilir. Kedi, Köpek, Kuş, Balık gibi bir çok hayvan türünden bahsedilebilir. Her türün kendine ait değişik özellikleri olabilir. Dolayısıyla her biri için değişik sınıfların tasarlanması gerekebilir.

# Kalıtım (Inheritance)

- Fakat ortak bir takım özelliklerinin olması da kaçınılmazdır ve her biri için bağımsız sınıflar tasarlandığında bu benzerlikler her birisi için tekrarlanmak durumunda kalacaktır . Bu yüzden önce tüm hayvanlar için bir sınıf oluşturulup diğer kedi,  köpek  gibi sınıflar bu sınıfın devamı gibi tasarlanabilir. İşte temelde bir sınıf tanımlanıp diğer sınıfları bu sınıftan türeterek özelleştirmeye kalıtım yoluyla türetme adı verilir.

- Türetme yapmak için sınıf tanımlaması şu şekilde yapılmalıdır:

```
class TüretilenSınıf : TemelSınıf
```

- Türetme işleminden sonra türetilen sınıf temel sınıfın bütün özelliklerine sahip olur.

# Kalıtım (Inheritence)

- Tip güvenliği olan dillerde farklı türdeki nesnelerin birbirine atanması istisna durumlar dışında yasaktır.

- Bu istisna durumlardan biri de türemiş sınıfın referansının temel sınıfa ilişkin bir referansa atanabilmesidir.

- Bu durumda temel sınıf türeyen sınıfın tüm özelliklerine erişemeyecek olmasına rağmen atama işlemi yapılabilmektedir.

Örneğin:

Bir **Hayvan sınıfı** oluşturulsun; her hayvanın boy, ağırlık gibi fiziksel özellikleri olsun. Ardından bu sınıftan bir **Kedi sınıfı** tanımlayalım. Hayvan sınıfında boy ve ağırlığı gösteren **OzellikGoster()** isimli bir metot olsun. Kedi sınıfında da hayvanın kedi olduğunu gösteren string tipinde özel bir değişken tanımlı olsun.

# Kalıtım (Inheritence)

```
class Hayvan //temel sınıf
{
 public double boy;
 public double agirlik;
 public void OzellikGoster()
 {
  Console.WriteLine("Boy="+boy);
  Console.WriteLine("Agirlik="+agirlik);
 }
}
```

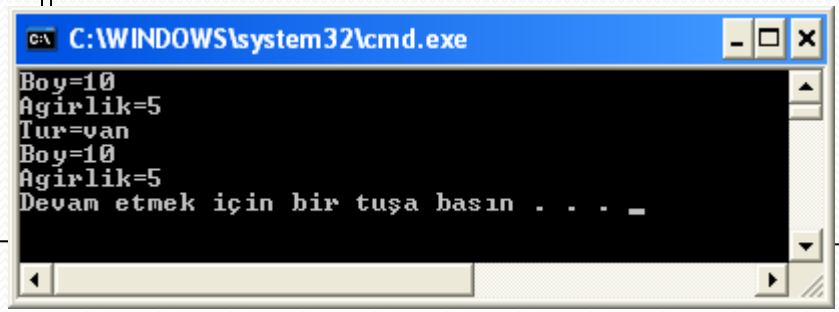
- Ayrıca Kedi sınıfında kedinin türünü yazacak bir de metot olsun. Türetme aşağıdaki şekilde gerçekleştirilir;

```
class Kedi : Hayvan//türetilmiş sınıf
   {
        public string Turu;
        public void TurGoster()
        {Console.WriteLine("Tur="+Turu);}
   }
```

- Burada Kedi türetilmiş(derived) sınıf, Hayvan da temel (based) sınıftır.

# Kalıtım (Inheritence)

```
class MainMetot
{
 static void Main()
 {
   Kedi k1= new Kedi();
   k1.agirlik=5;
   k1.boy=10;
   k1.Turu="van";
   k1.OzellikGoster();
   k1.TurGoster();
        Hayvan h1= new Hayvan();
        h1.agirlik=5;
        h1.boy=10;
        //h1.Turu="van"; //HATA!!!
        h1.OzellikGoster();
        //h1.TurGoster(); //HATA!!!
      }
}
```



# Kalıtım (Inheritence)

- Kalıtım yolu ile public ve protected elemanlar aktarılır. Diğer sınıfların kullanımına kapalı ancak türetme ile türemiş sınıfa geçebilen özellikler protected anahtar sözcüğü kullanılır.

- Eğer türetme söz konusu değilse protected olarak bilinen elemanlarla private olanlar arasında bir fark olmayacaktır.

- private özelliklere türetilen sınıflardan erişilemez.
- protected özellikler ise türeyen sınıfa private olarak geçer.

# Kalıtım (Inheritence)

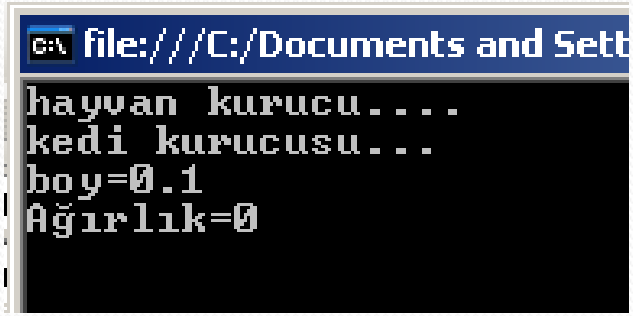
- C#'ta yapıcı metotların türetimiyle ilgili şu kurallar geçerlidir:
  - C#'ta yapıcı metotlar fiziksel olarak türetilmez.
  - Türemiş sınıf türünden bir nesne yaratıldığında önce ana sınıfın parametre almayan yapıcı metodu, ardından türemiş sınıftaki imzaya uyan yapıcı metot çalıştırılır.
  - Türemiş sınıf türünden nesne yaratımında daima türemiş sınıfın imzaya uyan bir yapıcı metodu olması gerekir.
  - Türemiş sınıf türünden nesne yaratımlarında, ana sınıfın parametre almayan yapıcı metodu yavru sınıfın üye elemanlarıyla işlem yapar.
  - Türemiş  sınıf türünden nesne yaratımında, Türemiş sınıftaki ilgili yapıcı metoda **base** takısı eklenmişse ana sınıfın parametre almayan yapıcı metodu çalıştırılmaz.

# Kalıtım (Inheritence)

- public  Hayvan()
- {
- Console.WriteLine("hayvan  kurucu....");
- Boy = 0.1;  Agirlik = 0.2;
- } // temel sınıf yapıcısı
- public Kedi()
- {
- Agirlik = 0;
- Console.WriteLine("kedi kurucusu...");
- } // Türmeiş sınıf yapısıcısı

- Main içinde tanımlı satırlar;
  - **Kedi k1 = new Kedi();**
  - **k1.OzellikGoster();**
  - **Console.ReadKey();**

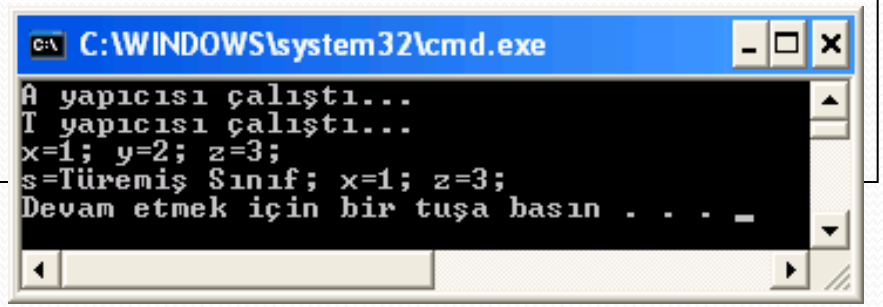
**Ekran çıktısı;**




İlk önce temel kurucu çalıştı, boy ve ağırlık değerlerini 0.1 ve 0.2 değerlerine eşitledi, ardından türemişin kurucusu çalıştı ve ağırlık değerini 0 yaptı, boy değerini ise türemiş kurucuda herhangi bir işlem yapılmadığı için temel sınıfın kurucu değerini kullanmış oldu.

# Kalıtım (Inheritence)

```
using System;
class A
{  public int x;  private int y;  protected int z;
    public A()
    {   x = 1;    y = 2;    z = 3;
    Console.WriteLine ("A yapıcısı çalıştı..."); }
   public void Listele() {
Console.WriteLine("x={0}; y={1}; z={2};", x,
y, z );  }
}
class T : A
{ public string s;
  public T()
  {s = "Türemiş Sınıf";
Console.WriteLine("T yapıcısı çalıştı...");
 }
public void Yaz()
 { Console.WriteLine("s={0}; x={1};
z={2};", s, x, z);
 }
}
class Program
{  static void Main()
   {  T t = new T();
       t.Listele();
       t.Yaz();
   }
}
```



# Kalıtım (Inheritence)

Programdaki hatayı bulunuz.

```
using System;
class ana
{
public ana(int a){}
}

class yavru:ana { }

class esas
{
static void Main() { yavru y=new yavru(); }
}
```

Bu program hata verir. ana sınıfta parametre almayan bir yapıcı metot yoktur. Ana sınıfta bir yapıcı metot tanımlandığı için varsayılan yapıcı metot oluşturulmamıştır.

# Kalıtım (Inheritence)

- base anahtar sözcüğü
- Yapıcı metotlar aşırı yüklenmişse türemiş sınıfın yapıcı metotları çağrılırken belli değerlerle temel sınıfında yapıcı metodunun çağrılması mümkündür ve bu işlem base anahtar sözcüğü ile yapılır.
- base anahtar sözcüğü yalnızca yapıcı metotlarla kullanılabilir. Yani base anahtar sözcüğünü yalnızca türemiş sınıftaki yapıcı metoda ekleyebiliriz ve base anahtar sözcüğünün ana sınıfta var olan bir yapıcı metodu belirtmesi gerekir.

  - Public T(string s, int x, int z): base (int x, int z)

# Örnekler

```
using System;
class A {
public int Ozellik1;
public int Ozellik2;
public A() { Console.WriteLine("Deneme"); }
public A(int ozellik1,int ozellik2) {
Ozellik1=ozellik1; Ozellik2=ozellik2; } }
 class B:A {
public int Ozellik3;
public int Ozellik4;
public B( int ozellik3,int ozellik4, int
ozellik1,int ozellik2): base(ozellik1,ozellik2)
{ Ozellik3=ozellik3; Ozellik4=ozellik4; } }
```

```
class esas {
static void Main() {
B b=new B(3,4,1,2);
Console.WriteLine (b.Ozellik1+"
"+b.Ozellik2+" "+b.Ozellik3+"
"+b.Ozellik4);
} }
```

- Bu program ekrana 1 2 3 4 yazar. Bu örnekte base anahtar sözcüğü ana sınıftaki yapıcı metodu temsil etmektedir. Türemiş sınıfta base sadece ana sınıftaki değer alan yapıcıya gittiğinden ana sınıfın boş yapıcı metodu çalışmaz

# Çoklu kalıtım (türetme)

- Sınıflar, ard arda türetilebilir. Yani örneğin B sınıfı A sınıfında türetilip C sınıfı da B sınıfından türetilebilir.

- Bu durumda C sınıfı türünden bir nesne yarattığımızda eğer C sınıfının ilgili yapıcı metoduna base takısını eklememişsek önce A, sonra B, sonra da C sınıfının yapıcı metotları çalıştırılır. Yani gidişat temel sınıftan türemiş sınıfa doğrudur.

- C sınıfı hem A'nın hem de B'nin bütün üye elemanlarına sahip olur. Örnek:

# Çoklu kalıtım (türetme)

```
using System;
class A {
public A()
{ Console.WriteLine("A sınıfı"); } }
class B:A {
public B() { Console.WriteLine("B sınıfı"); } }
class C:B {
public C() { Console.WriteLine("C sınıfı"); }
static void Main()
{ C nesne=new C(); } }
```

```
A sınıfı
B sınıfı
C sınıfı
```

# Çoklu kalıtım (türetme)

- Örnek

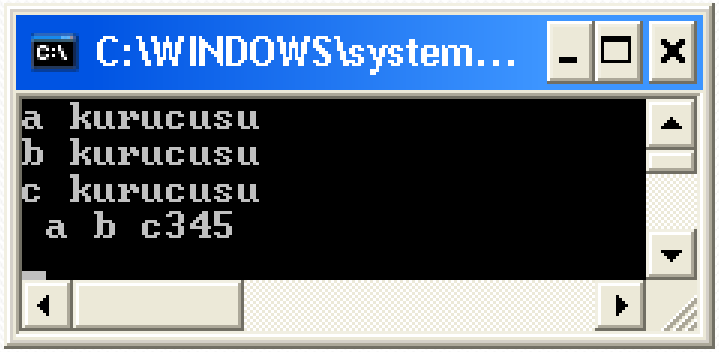
```
class A
 {
   public int a;
   public A(int a)
   {
   Console.WriteLine("a kurucusu");
    this.a =a;
        }
    }
```

```
class B : A
    {
        public int b;
        public B(int a,int b) : base(a)
        {
            Console.WriteLine("b kurucusu");
            this.b = b;
        }
    }
```

# Çoklu kalıtım (türetme)

```
class C : B
 {
    public int c;
    public void yaz()
    {
        Console.WriteLine(" a b c" +a+b+c );
    }
    public C(int a,int b,int c) : base(a,b)
    {
        Console.WriteLine("c kurucusu");
        this.c = c;
    }
 }
```

```
class deneme {

        static void Main(string[]
       args)
       {
           C cc = new C(3,4,5);
           cc.yaz();
           Console.ReadKey();
       }
}
```



# Çoklu kalıtım (türetme) Örnek

```
using System;
class A
{  public int OzellikA;
   public A(int a)  {    OzellikA=a;  }  }
class B:A
{ public int OzellikB;
   public B (int b)  {    OzellikB=b;  }
 }
class C:B
{  public int OzellikC;
   public C(int c,int b):base(b)
   {    OzellikC=c;  }
   static void Main()
   {   C nesne=new C(12,56);
 Console.WriteLine( nesne.OzellikA+"
"+nesne.OzellikB+" "+nesne.OzellikC);
   }
}
// Hata verir
```

- C nesnesi oluşturulurken base ile değeri B sınıfa gönderdi. B sınıfının yapıcı metodu çalıştırıldı fakat B sınıfının yapıcı metodunda base takısı olmadığı için B sınıfına göre ana sınıfın (A sınıfı) parametre almayan yapıcı metodu çalıştırılmaya çalışılmıştır. A sınıfının parametre almayan yapıcı metodu olmadığı için program hata vermiştir.
- Çözüm: public B (int b):base (b);
- B sınıfı değer göndermeyecek ise A ve B sınıflarında varsayılan yapıcılar da tanımlanacak.
- public A() {}
- public B() {}

# İsim Saklama (Name Hiding)

- Türemiş sınıfta bazen temel sınıftaki üye elemanla aynı isimli bir eleman tanımlanmış olabilir. Bu durumda temel sınıftaki elemana normal yollarda erişmek mümkün değildir çünkü türeyen sınıftaki eleman temel sınıftaki elemanı gizlemiştir.

- Temel sınıftaki elemana erişmek için yine base anahtar sözcüğünden faydalanılır. Base ile hem özelliklere hem de metotlara erişilebilir.

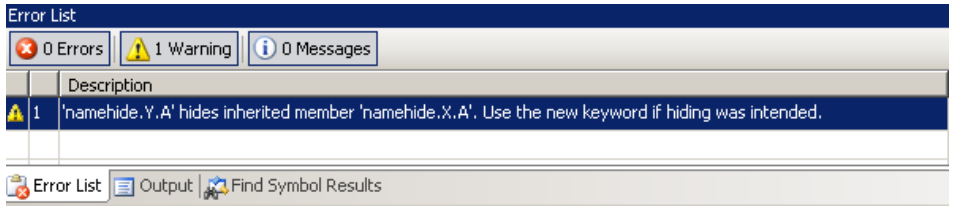
- base anahtar sözcüğünün bu şekildeki gibi kullanımı this referansına benzemektedir.

- this referansı kendisini çağıran sınıfı temsil ederken base anahtar sözcüğü türetmenin yapıldığı temel sınıfı temsil eder.

# İsim Saklama (Name Hiding)

```
class X
{    protected int a;
     //public X() { }
      public X(int a)  {   this.a=a;  }
      public int A //dikkat
      { get  {  Console.WriteLine("X");  return a;  } }
 }
class Y: X
{
    protected int b;
    public Y(int b) :base(b) { this.b=base.a;  }
   public int A // dikkat- public new int A
   {
     get  { Console.WriteLine("Y sınıfı");  return b; }
    }
}
class Program
{ static void Main()
{   Y y=new Y(5);     int deneme=y.A;
} }
```


- Program uyarısı;

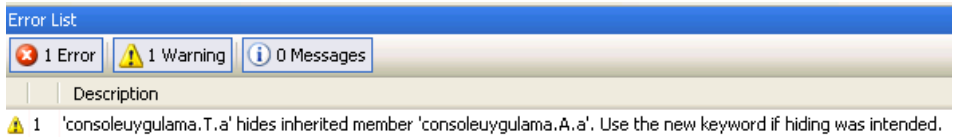



- (Uyarı) Y.A elemanı X.A elemanını gizlemiştir.
- Bu yüzden burada **new** anahtarı kullanılabilir. Her zaman isim gizlenmesi yaparken isim gizlemeyi açıkça belirtmek için temel sınıftaki bir elemanı gizlerken türemiş sınıftaki elemanın bildirimine **new** anahtarı eklemeliyiz. Bu sayede uyarıyı almayı da engellemiş oluruz.
- base(b) parametresi olmasaydı // public X() {} yapıcı metodunu tanımlamak zorundaydık. X Parametreli yapıcısı olduğundan varsayılan yapıcı metodu Y sınıfı çağıramazdı.

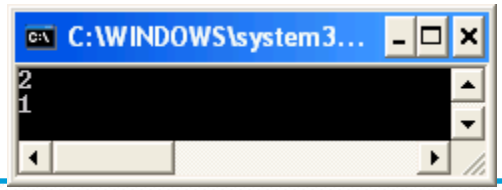
# İsim Saklama (Name Hiding)

## Örnek

```
using System;
class A
{   public int a;
    public A()   {  a = 1; }
}
class T : A
{   public int a;
  public T()  { a = 2;  }
}
 class Program
 {
   static void Main()
  {
   T t = new T();
  Console.WriteLine(t.a);
   }
}
```

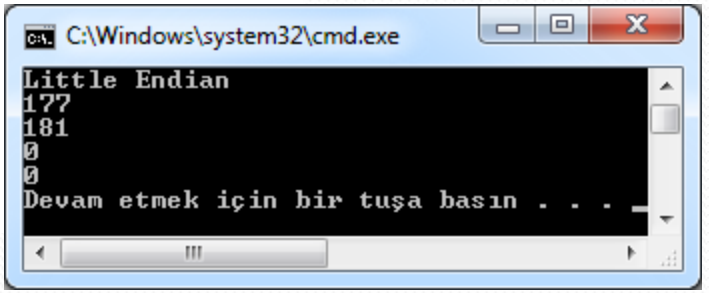

```
using System;
class A
{ public int a;
  public A()  {  a = 1;    }
}
class T : A
{
  public new int a;
  public int b ;
  public T()   { a = 2;    b = base.a;    }
}
class Program
{
  static void Main()
  {
  T t = new T();
  Console.WriteLine(t.a);
  Console.WriteLine(t.b);
  Console.ReadLine();
  }
}
```



# Visual Studio.Net -C#
# 7. HAFTA
# Sanal Metotlar, Özet Sınıflar ve Arayüzler, İstisnai Durumlar, Temsilciler ve Olaylar, Şablon Tipler

# System.BitConverter

```
using System;

class Program
{
    static void Main()
    {
        if (BitConverter.IsLittleEndian)
            Console.WriteLine("Little Endian");
        else
            Console.WriteLine("Big Endian");

        int a = 46513;   // 256^0*177+256^1*181+256^2*0+256^3*0=46513

        byte[] b = BitConverter.GetBytes(a);

        foreach (byte x in b)
            Console.WriteLine(x);
    }
}
```



# System.Buffer Örnek

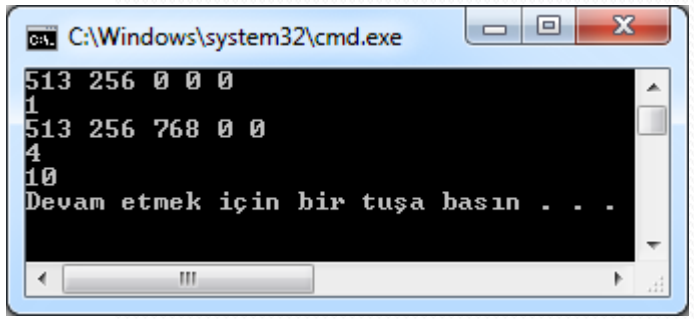
- Short (2 byte)
- 0.byte 1.byte 2.byte 3.byte 4.byte
- 1 2*256 0 1*256 0

- 5.byte // (SetByte ile 5. byte ekleme)
- 3*256

```
using System;

class Program
{
    static void Main()
    {
        byte[] kaynak = { 1, 2, 0, 1 };
        short[] hedef = new short[5];

        Buffer.BlockCopy(kaynak, 0, hedef, 0, 4);

        foreach (short s in hedef)
            Console.Write(s + " ");

        Console.WriteLine("\n" + Buffer.GetByte(hedef, 0));
        Buffer.SetByte(hedef, 5, 3);

        foreach (short s in hedef)
            Console.Write(s + " ");

        Console.WriteLine();
        Console.WriteLine(Buffer.ByteLength(kaynak));
        Console.WriteLine(Buffer.ByteLength(hedef));
    }
}
```

Byte 8 bit
Short 16 bit

$2*256^1+1*256^0=513$
$1*256^1+0*256^0=256$
$3*256^1+0*256^0=768$



# Örnekler

```
class Program
    {
        static void Main(string[] args)
        {
            model1 oto1 = new model1();
            model2 oto2 = new model2();
            oto1.tur = "Sedan";
            oto1.silindir_sayisi = 4;
            oto1.subap_sayisi = 8;
            oto1.guc = 75;
            oto1.tork = 100;
            oto1.ozellikyaz();
            Console.WriteLine("********");
            oto2.model2_boy = 6;
            oto2.model2_agirlik = 900;
            oto2.model2_renk = "Kırmızı";
            oto2.ozellikyaz();
            Console.WriteLine("********");
            oto2.goster();
            Console.WriteLine("********");
            oto1.goster();
            Console.ReadLine();

        }
    }
}
```

# Örnekler:

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

namespace ConsoleApplication9
{
    class oto
    {
        protected double boy=5;
        protected double agirlik=800;
        protected string renk="Sarı";
        public void goster()
        {
            Console.WriteLine("Boy=" + boy);
            Console.WriteLine("Ağırlık=" + agirlik);
            Console.WriteLine("Renk=" + renk);
        }
    }
    class model1:oto
    {
        public string tur;
        public int silindir_sayisi;
        public int subap_sayisi;
        public int tork  ;
        public int guc ;
        public void ozellikyaz()
        {
  Console.WriteLine("Tür=" + tur);
  Console.WriteLine("Boy="+boy);
  Console.WriteLine("Ağırlık=" + agirlik);
  Console.WriteLine("Renk=" + renk);
  Console.WriteLine("Silindir Sayisi=" + silindir_sayisi);
  Console.WriteLine("Subap Sayısı=" + subap_sayisi);
  Console.WriteLine("Tork=" + tork);
  Console.WriteLine("Güç=" + guc);
        }
    }
```

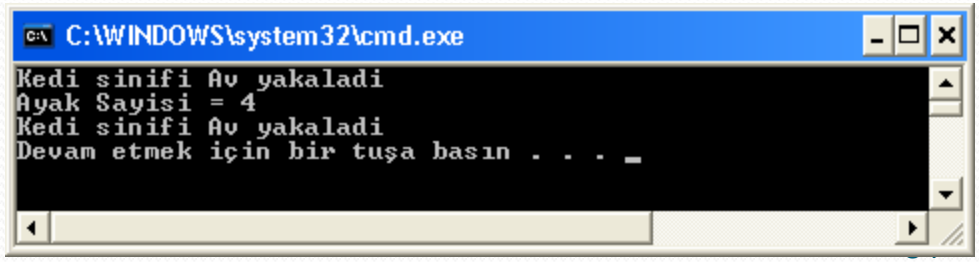
```
class model2 : oto
    {
        public double model2_boy
        {
            get { return boy; }
            set { boy = value; }
        }
        public double model2_agirlik
        {
            get { return agirlik; }
            set { agirlik = value; }
        }
        public string model2_renk
        {
            get { return renk; }
            set { renk = value; }
        }
        public string tur="Hatchback";
        public int silindir_sayisi=8;
        public int subap_sayisi=16;
        public int tork=300;
        public int guc=210;
        public void ozellikyaz()
        {
          Console.WriteLine("Tür=" + tur);
          Console.WriteLine("Boy=" + model2_boy);
          Console.WriteLine("Ağırlık=" + model2_agirlik);
          Console.WriteLine("Renk=" + model2_renk);
          Console.WriteLine("Silindir Sayisi=" + silindir_sayisi);
          Console.WriteLine("Subap Sayısı=" + subap_sayisi);
          Console.WriteLine("Tork=" + tork);
          Console.WriteLine("Güç=" + guc);
        }
    }
```

# Örnek:

```
using System;

class A
{
    public int x;
    private int y;
    protected int z;

    public A()
    {
        x = 1;
        y = 2;
        z = 3;
        Console.WriteLine("A yapıcısı çalıştı...");
    }

    public void Listele()
    {
        Console.WriteLine("x={0}; y={1}; z={2};",x,y,z);
    }
}

class T : A
{
    public string s;

    public T()
    {
        s = "Türemiş Sınıf";
        Console.WriteLine("T yapıcısı çalıştı...");
    }

    public void Yaz()
    {
        Console.WriteLine("s={0}; x={1}; z={2};", s, x, z);
    }
}

class Program
{
    static void Main()
    {
        T t = new T();
        t.Listele();
        t.Yaz();
    }
}
```

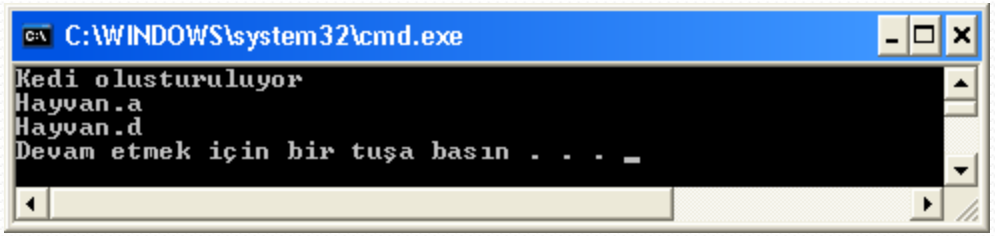
# Örnek

```
using System;
class Kedi {
   protected int ayakSayisi = 4 ;
   public void yakalaAv()    {
      Console.WriteLine("Kedi sinifi Av yakaladi");    }
 }
class Kaplan : Kedi {
   public Kaplan()    {
      Console.WriteLine("Ayak Sayisi = " + ayakSayisi);    }
 }
class deneme {
public static void Main(string[] args)    {
      Kedi kd= new Kedi() ;
      kd.yakalaAv();
            Kaplan kp = new Kaplan();
      kp.yakalaAv();    }
}
```



# Örnek

```
using System;
public class Hayvan {
        protected String a = "Hayvan.a";
        String b = "Hayvan.b"; //friendly
        private String c = "Hayvan.c";
        public String d = "Hayvan.d"; }
public class Kedi:Hayvan { // Türeyen
  public Kedi() {
        Console.WriteLine("Kedi olusturuluyor");
        Console.WriteLine(a);
        //Console.WriteLine(b); // ! Hata ! erisemez ?
        //Console.WriteLine(c); // ! Hata ! erisemez ?
        Console.WriteLine(d);
  } }
class deneme {
public static void Main(string[] args)  {
        Kedi k = new Kedi(); }
}
```



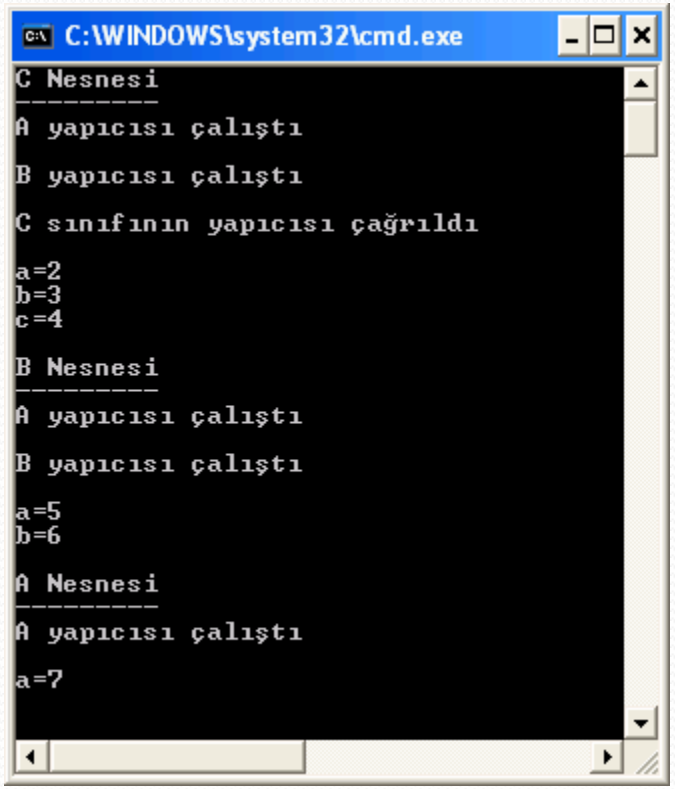
# Örnek

```
using System;
class A
{
    public int a;
    public A(int a)
    {
this.a = a;
Console.WriteLine("A yapıcısı çalıştı\n");
    }

}
class B : A
{
    public int b;
    public B(int a,int b):base(a)
    {
this.b=b;
Console.WriteLine("B yapıcısı çalıştı\n");
    }

}
class C:B
{   public int c;
    public C(int a,int b, int c):base(a, b)
    {
this.c = c;
Console.WriteLine("C sınıfının yapıcısı çağrıldı\n");
    }
}
```

```
class Program
{
    static void Main()
    {

        Console.WriteLine("C Nesnesi");
        Console.WriteLine("---------");
        C c = new C(2, 3, 4);
        Console.WriteLine("a="+c.a);
        Console.WriteLine("b=" + c.b);
        Console.WriteLine("c=" + c.c+"\n");

        Console.WriteLine("B Nesnesi");
        Console.WriteLine("---------");
        B b = new B(5, 6);
        Console.WriteLine("a=" + b.a);
        Console.WriteLine("b=" + b.b+"\n");

        Console.WriteLine("A Nesnesi");
        Console.WriteLine("---------");
        A a = new A(7);
        Console.WriteLine("a=" + a.a+"\n");
        Console.ReadLine();
    }
}
```

# Örnek

- Yukarıdaki programda, base anahtar sözcüğü sınıf hiyerarşisinin en tepesindeki sınıfı temsil etmektedir. C sınıfında base anahtar sözcüğü B sınıfı anlamına gelirken, B sınıfında base anahtar sözcüğü A sınıfı anlamına gelmektedir.



# Örnek

```
class Program
    {
        public static void Goster(oto Oto)
        {
            Console.WriteLine(Oto.Tur); //Hata ulaşamaz
            Console.WriteLine(Oto.MotorGucu);
            Console.WriteLine(Oto.Tork);
            Console.WriteLine(Oto.Renk);
        }

        static void Main(string[] args)
        {
            oto oto1=new oto
    (75,100,"Kırmızı");
            Goster(oto1);
            Console.WriteLine("----------");
            model1 oto2=new
    model1("Fiat",100,110,"Beyaz");
            Goster(oto2);
            Console.WriteLine("----------");
            model2 oto3=new model2
    ("Renault",100,120,"Siyah");
            Goster(oto3);
            Console.ReadLine();

        }
    }
}
```

# Örnek

```
using System;
using System.Collections.Generic;
using System.Text;
namespace ConsoleApplication9
{
    class oto
    {
        protected double motorgucu=5;
        protected double tork=800;
        protected string renk="Sarı";
        public oto(double guc, double tork, string
    renk)
        {
            this.motorgucu = guc;
            this.tork = tork;
            this.renk = renk;
        }
        public void ozellikgoster()
        {
            Console.WriteLine("Motor Gücü=" +
    motorgucu);
            Console.WriteLine("Tork=" + tork);
            Console.WriteLine("Renk=" + renk);
        }
        public double MotorGucu
        {
            get { return motorgucu; }
            set { motorgucu = value; }
        }
        public double Tork
        {
            get { return tork; }
            set { tork = value; }
        }
        public string Renk
        {
            get { return renk; }
            set { renk = value; }
        }
    }
```

```
class model1:oto
    {
        public string Tur;
        public model1(string tur,double guc,
    double tork,string renk):base
    (guc,tork,renk)
        {
            this.Tur=tur;
        }
        public void TurGoster()
        {
            Console.WriteLine("Türü"+Tur);
        }
    }

    class model2 : oto
    {
        public string Tur;
public model2(string tur,double guc, double
    tork,string renk):base (guc,tork,renk)
        {
            this.Tur=tur;
        }
        public void TurGoster()
        {
            Console.WriteLine("Türü"+Tur);
        }
    }
```

# İsim Saklama (Name Hiding)

```
using System;
  class X { protected int a;
         public X(int a)   {
         Console.WriteLine("X " + a);
        this.a=a;   }
      public int A {
        get {
        Console.WriteLine("X Sınıfı="+a);
        return a;  }
        } }
  class Y: X  {
     protected int b;
     public Y(int a):base(a)  {
        Console.WriteLine("Y " + a);
        this.b=a;  }
     new public int A {
        get {
        Console.WriteLine("Y sınıfı="+b);
        return b;  }
     }   }
```

```
class tarih
{
   static void Main()
   {
      Y y=new Y(5);
      X yy = new X(6);

      int deneme=y.A;
      int deneme2=yy.A;
   }
}


/// işleminin sonucunu söyleyiniz
```

# İsim Saklama (Name Hiding)

- Y y=new Y(5);
- Öncelikle Ana sınıfın yapılandırıcısı çalıştı
- Daha sonra türemiş sınıfın yapılandırıcısı çalıştı
- X yy = new X(6);
- Sadece Ana sınıfın yapılandırıcısı çalıştı
- int deneme=y.A;
- Türemiş sınıfıa ait get bloğu çalıştı
- int deneme2=yy.A;
- Ana sınıfa ait get bloğu çalıştı