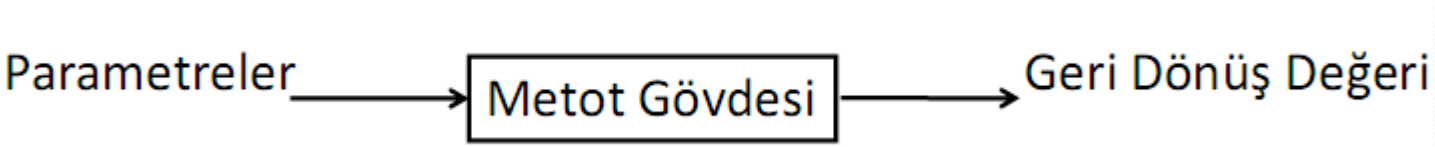
Visual Studio.Net -C#
4. HAFTA
Metotlar,
get-set yöntemi

Metot Kavramı

- Programın herhangi bir yerinde kullanmak için belirli bir işi yerine getirmek amacıyla tasarlanmış alt programlara **metot** denir.
- Metotlar tek başlarına çalıştırılabilen yapılar değildir.
- Metodun iş yapması için kendisini çağıran metottan aldığı bilgilere parametre(argüman), kendisini çağıran fonksiyona döndürdüğü bilgiye ise geri dönüş değeri (return value)



Metot Bildirimi

- C#'ta bildirilen bütün metotlar mutlaka bir sınıfın içinde olmalıdır. Bir sınıfın üyesi olmayan metotlar bildirilemez. Metotları tanımlarken kullanılan bildirim şu şekildedir:

- **[erişim belirteçleri]** **<dönüş değeri>** **metot ismi** **(parametre listesi)**
  { metot gövdesi }

- Örn:
- `int MetotAdi(int a,int b)`
- `{ return a+b; }`
- Erişim belirteçleri: metoda nerelerden erişilebileceğini ayarlar (public, private…) Belirtilmediği durumlarda "private" olarak kabul edilir. Yani sadece tanımlandığı sınıf içinde kullanılabilen bir metot olur.

Metot Bildirimi

- C#'ta bir metot kullanılacaksa metodun içinde bulunduğu sınıf türünden bir nesne tanımlanır ve "." operatörü ile metot çağrılır.

```
using System;
class Metotlar
{
    int Topla(int a, int b) { return a + b; }

    static void Main()
    {
        Metotlar nesne = new Metotlar();
        int a = nesne.Topla(2, 5);
        Console.Write(a);
    }
}
```

Metot Bildirimi

- static olarak tanımlanan metotlara erişmek için metodun içinde bulunduğu sınıf türünden bir nesne yaratmaya gerek yoktur. static olarak tanımlanan metotlara sadece metodun adını yazarak erişilebilir. Örnek:

```
using System;
class Metotlar
{
    static int Topla(int a, int b)
    { return a + b; }
    static void Main()
    {
        int a = Topla(2, 5);
        Console.Write(a);
    }
}
```

Metot Bildirimi

- Eğer metot, içinde bulunduğumuz sınıfta değil de, başka bir sınıf içinde oluşturulmuş ise o metodu kullanabilmek için önce sınıfı yazmamız gerekir. Örnek:

```
using System;
class Metotlar1
 {    public static int Topla(int a,int b) {  return a+b; }    }
class Metotlar2
{  static void Main()
  {
    int a=Metotlar1.Topla(2,5); //nesne oluşturulmadı
   Console.Write(a);
  }
}
```

Metot Bildirimi

- static olmayan bir metodu başka bir sınıf içinde kullanma

```
using System;
class Metotlar1
 {
   public  int Topla(int a,int b)
   {  return a+b; }
 }
class Metotlar2
{
  static void Main()
  {
   Metotlar1 nesne=new Metotlar1(); //nesne oluşturuldu
   int a=nesne.Topla(3,9);
   Console.Write(a);
  }
}
```

Metot Bildirimi

- **Metotlarla ilgili önemli özellikler**

- **1-** Metotlar isimlendirilirken, değişken isimlendirmede uyulması gereken kurallara uyulmalıdır. Ayrıca Main() isimli ikinci bir metot daha tanımlanamaz.

- **2-** Eğer bir metot geriye değer döndürmeyecekse "**void**" olarak tanımlanır. Eğer giriş parametresi yoksa parantez içi boş bırakılır.
- (C ve C++ dillerinde bir fonksiyonun parametre almadığını açıkça bildirmek için parametre parantezine void anahtar sözcüğü yazılır, fakat C#'da böyle bir kullanım yoktur.)

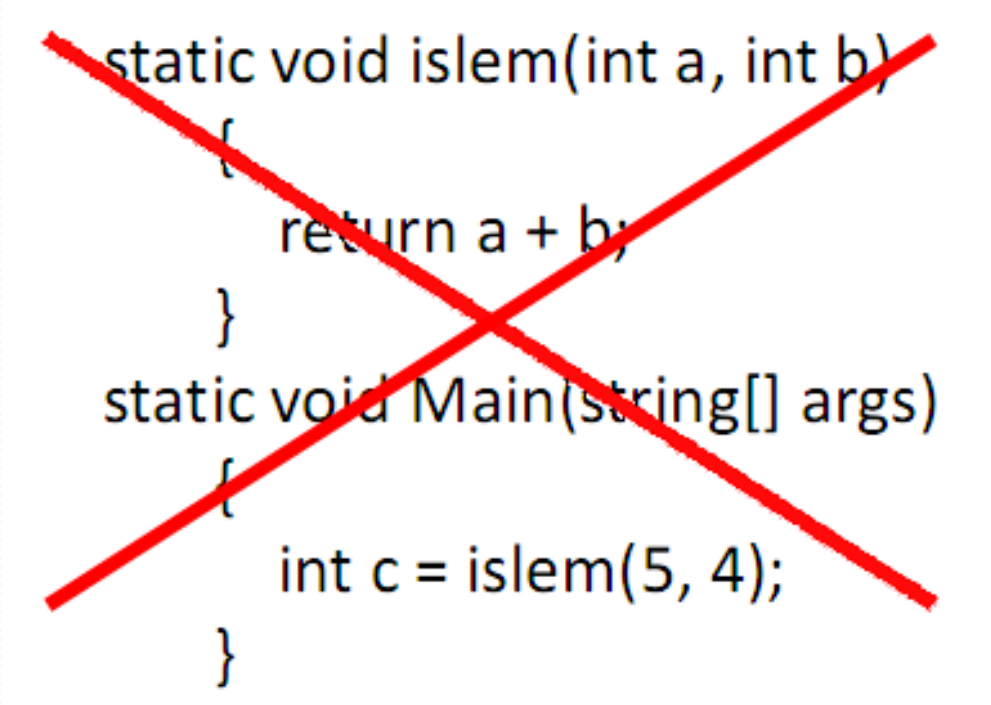
- **3-** Geri dönüş değeri olmayan metotlarda (void ile belirtilmiş) **return** anahtar sözcüğünün yanında herhangi bir ifade olmadan da kullanılabilir.

Metot Bildirimi

- **Örnek:**

```
using System;
class Metotlar1
{
 static void Yaz(object a,int b)
  {
     if(b>10)  return; /* değer tutmayan return komutu kullanımı sadece çıkışı sağlar */
for(;b>0;b--)
       Console.Write(a);
 }
static void Main()
{    Yaz("Merhaba",10); }
}
```

Metot Özellikleri

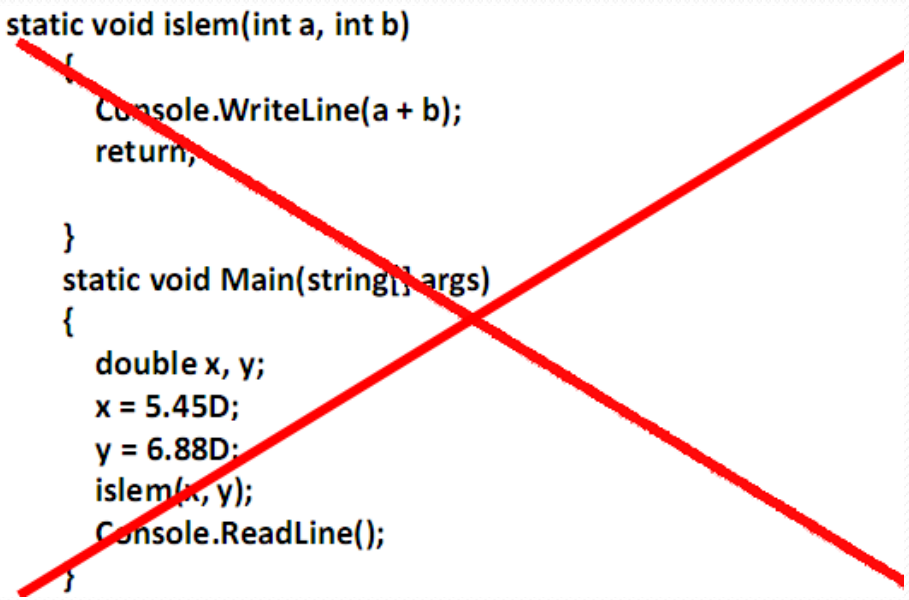


- Geri dönüş değeri olmayan metotlarda return anahtar sözcüğünü bir ifade ile kullanmak yasaktır

Metot Bildirimi

- **Metotlarla ilgili önemli özellikler**
- **4-**“**return**” anahtar sözcüğü ile geri değer döndürürken de türlerin uyumlu olmasına dikkat edilmelidir.

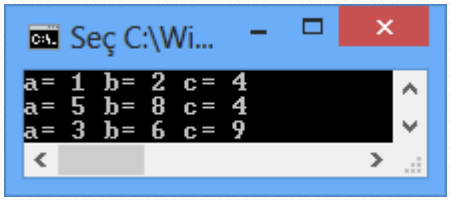
- **5-** Metotlara parametre gönderirken tür dönüşümü kurallarına uyulmalıdır, uygun türler kullanılmalıdır. Geri dönüş değerinde yapılan gizli tür dönüşümleri ve tür uyumsuzluk kuralı metot parametreleri için de geçerlidir.



Metot Bildirimi

- **Metotlarla ilgili önemli özellikler**
- **6-** Metot çağırırken parametreler eksiksiz girilmelidir. Eksik parametreli metot çağrıları geçersizdir. Eğer parametreler daha önce tanımlıysa eksik girilebilir. Ama bir parametrenin değeri atandıktan sonra bundan sonraki diğer parametrelerin değeri de atanmalıdır.

```
class Program
{    public static void metotlar(int a, int b = 2, int c=4)
    { Console.WriteLine("a= " + a + " b= " + b + " c= " + c);
    }
    static void Main(string[] args)
    {
        metotlar(1); metotlar(5,8); metotlar(3,6,9);
    }
}
public static void metotlar(int a, int b = 2,int c)
```



- Derleme hatası verir. **b** değeri belirlenmişse bundan sonraki tüm değerler belirlenmelidir.

Metot Bildirimi

- **Metotlarla ilgili önemli özellikler**
- **7-** Metotlar parametre almayabilirler.
- Örnek:

```
using System;
class Metotlar1
{
    static void Yaz()
    { Console.Write("deneme"); }
    static void Main()
    {
      Yaz();
    }
}
```

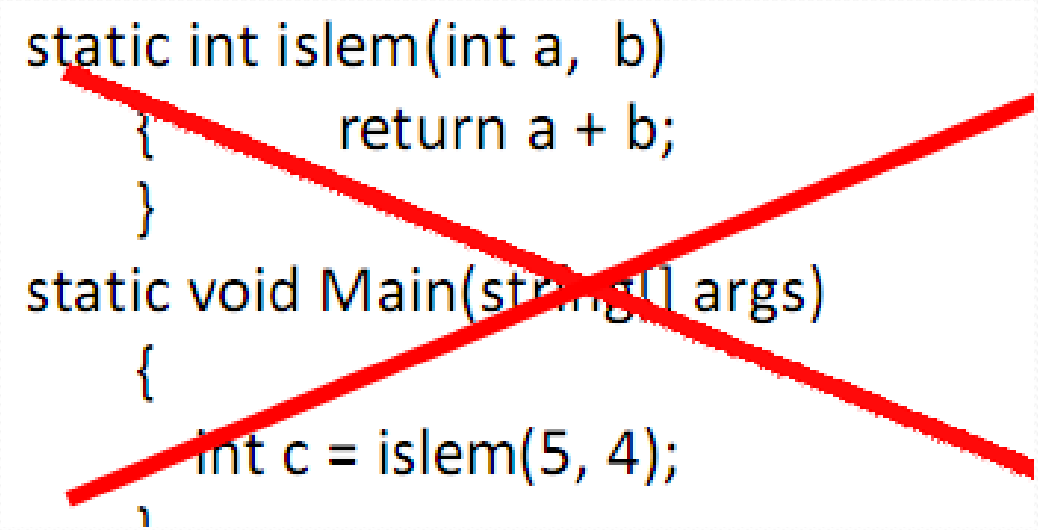
Metot Özellikleri

- **8-**Bir metot içerisinde başka bir metodun yapısı bildirilemez.

```
class Program
  {
    static void islem(double a, double b)
    {
      Console.WriteLine(a + b);
      return;
      int carp(int m,int n)
      { return m*n;}
    }
    static void Main(string[] args)
    {
      int x, y;
      x = 5;
      y = 6;
      islem(x, y);
      Console.ReadLine();
    }

  }
```

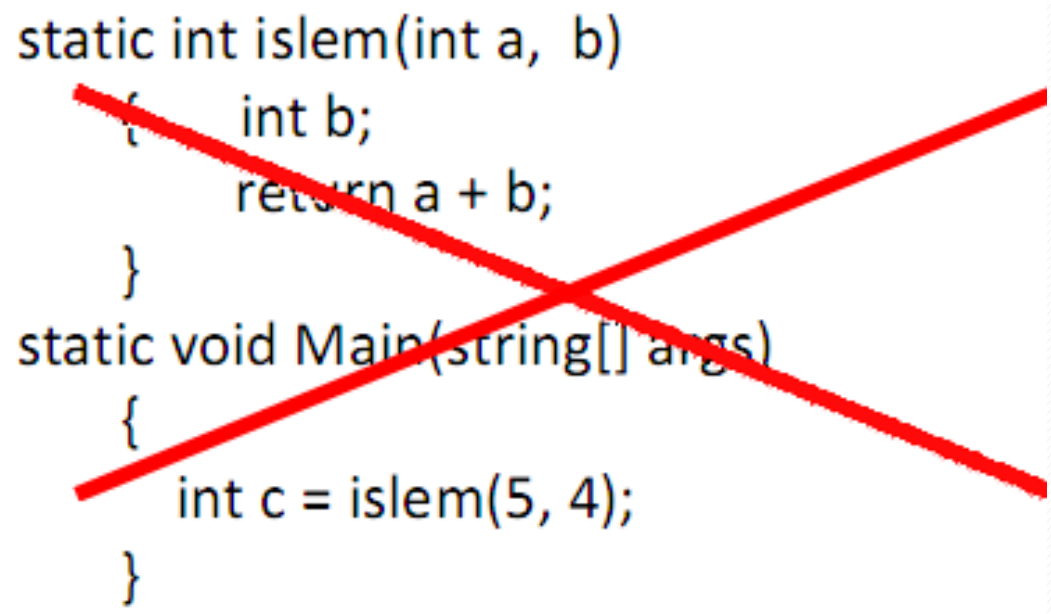
Metot Özellikleri

- **9-** Metot parametre parantezindeki değişkenlerin türlerinin tek tek belirtilmesi gerekir. ‘,’ ile ortak tür bildirimi yapılamaz.



Metot Özellikleri

- **10-**Metot parametrelerinin metot içerisinde tanımlanması geçersizdir.



Metot Özellikleri

- **11-** Metotlar içinde tanımlanan değişkenler programın başından sonuna kadar durmaz. Bu değişkenler, programın akışı metoda geldiğinde tanımlanır. Metodun işlevi bittiğinde ise bellekten silinirler.

- Programın akışı tekrar metoda geldiğinde değişkenler yeniden tanımlanır ve sonunda yine bellekten silinir. Bu tür değişkenlere otomatik ömürlü nesneler denir.

Metot Parametresi Olarak Diziler

Diziler C# ‘ta ayrı bir tür olarak ele alındığı için metotlara dizileri aktarmak normal veri türlerinde olduğu gibidir.

```
using System;
 class Metotlar1
 {
    static void Yaz(int[] dizi)
    {  foreach(int i in dizi)
        Console.WriteLine(i);
    }
static void Main() {
 int[] dizi={1,2,4,7,9}; Yaz(dizi); }
}
```

Metot Parametresi Olarak Diziler

Eğer yalnızca int[] türündeki değil bütün türlerdeki dizileri ekrana yazan genel bir metot yazmak istiyorsak:

```
using System;
 class Metotlar1
 {
    static void Yaz (Array dizi)
    {  foreach(object  i in dizi)
        Console.WriteLine(i);
    }
static void Main() {
 int [ ] dizi={1,2,4,7,9}; Yaz(dizi); }
}
```

Metot Parametresi Olarak Diziler

Dizilerde değişkenlerdeki gibi bir bilinçsiz tür dönüşümünden bahsetmek imkansızdır. Array yerine object kullanılsaydı hata verirdi.

```
using System;
 class Metotlar1
 {
    static void Yaz (object [] dizi)
    {  foreach(object  i in dizi)
        Console.WriteLine(i);
    }
static void Main() {
 int[] dizi={1,2,4,7,9}; Yaz(dizi); }
}
```

Değer ve Referans Parametreleri

- Referans türleri metotlara aktarılırken bit bit kopyalanmaz, yani metoda aktarılan sadece bir referanstır. Metot içinde yapılacak değişiklikler nesnenin(dizinin) değerini değiştirebilir.

- C# dilinde değer tipleri metoda bit bit kopyalanır.

- Değer tipleri parametre olarak verildiğinde fonksiyon içerisinde yeni bir nesne oluşturulur ve değeri buraya kopyalanır. Yani fonksiyon içerisinde değer tipindeki bir veri değiştiğinde çağıran programdaki ana değer değişmez.

Dizi ve değişken parametreler arasındaki fark

- Aşağıdaki iki programı karşılaştırın:

```
using System;
class Metotlar1 {
 static void Degistir(int[] dizi)
 { for(int i=0;i<5;i++)
   dizi[i]=10; }
static void Yaz(Array dizi)
 { foreach(object a in dizi)
Console.WriteLine(a); }
 static void Main() {
 int[] dizi={1,2,4,7,8}; Degistir(dizi);
Yaz(dizi); } }
```

```
using System;
class Metotlar1 {
static void Degistir(int sayi)
 { sayi=10; }
static void Yaz(int sayi)
 { Console.WriteLine(sayi);
}
 static void Main()
 { int sayi=1; Degistir(sayi); Yaz(sayi); }
}
```

Ref ve Out Anahtar sözcükleri

- Değer tiplerinin metoda referans tip olarak aktarılması için “ref” ve “out” anahtar sözcükleri kullanılır.

- **ref** anahtar sözcüğü ile belirtilen parametreler değer tipi de olsa referans olarak aktarılırlar. Genellikle değer tiplerini referans olarak metotlara geçirmeye zorlamak için kullanılırlar.

- Referans tipleri metotlara zaten referans olarak aktarıldıkları için ref anahtar sözcüğünü kullanmaya gerek yoktur. Fakat, kullanılması da kısıtlanmamıştır.

Ref ve Out Anahtar sözcükleri

Örnek:

```
using System;
class Metotlar1
{
    static void Degistir ( ref int sayi)     { sayi=10; }

    static void Yaz(int sayi)    { Console.WriteLine(sayi); }

    static void Main()
   { int sayi=1;       Degistir( ref sayi);       Yaz(sayi); }

 }
```

Ref ve Out Anahtar sözcükleri

- ref sözcüğü, hem metodu çağırırken , hem de metodu yaratırken değişkenden önce yazılması gerekiyor.

```
using System;

class RefOrnek
{
    static void DegerTipi(ref int x)
    {
        x = 111;
    }

    static void Main()
    {
        int x = 25;

        DegerTipi(x);
        Console.WriteLine(x);
    }
}
```

Ref ve Out Anahtar sözcükleri

- ref sözcüğüyle bir değişkenin metoda adres gösterme yoluyla aktarılabilmesi için esas programda değişkene bir ilk değer verilmelidir. Yoksa program hata verir.

```
using System;
class Metotlar1
{
  static void Degistir(ref int sayi)      {       sayi=10;   }
  static void Yaz(int sayi)   {       Console.WriteLine(sayi);   }
  static void Main()
  {    int sayi;
       Degistir(ref sayi);
       Yaz(sayi);
  }
}
```

Ref ve Out Anahtar sözcükleri

- İlk değer verilmediği durumlarda program hata verecektir. Bu tür hataları önlemek için **out** anahtar sözcüğünü kullanırız.

- out anahtar sözcüğünün ref anahtar sözcüğünden tek farkı ilk değer verme zorunluluğunun olmayışıdır. Değer vererek de kullanılır.

- **NOT:** ref sözcüğünün dizilerle kullanımı gereksiz olmasına rağmen C# bunu kısıtlamamıştır. Ancak out sözcüğü dizilerle kullanılamaz.

Ref ve Out Anahtar sözcükleri

```
using System;

class OutOrnek
{
    static void DegerTipi(out int x)
    {
        x = 111;
    }

    static void Main()
    {
        int x;

        DegerTipi(out x);
        Console.WriteLine(x);
    }
}
```

Metotların Aşırı Yüklenmesi (Method OverLoading)

- C#'ta parametre sayısı ve/veya parametrelerin türleri farklı olmak şartıyla aynı isimli birden fazla metot tanımlanmasına metotların aşırı yüklenmesi denir.

- Çalışma zamanında hangi metodun çağrıldığını belirleyebilmek için metodun imzasına (method signature) bakılır.

- Bir metodun imzası metodun adı, parametrelerin sayısı ve türleri ile ilgilidir. Metodun geri dönüş değerleri imzasına dahil değildir. Aynı parametrelere sahip geri dönüş değerleri farklı olan fonksiyonlar derleme anında hataya neden olurlar.

Örnek

```
using System;
class Metotyükleme
{
    static void Metot1 (int x, int y)
    { Console.WriteLine("1. Metot çağrıldı"); }

    static void Metot1 (float x, float y)
    { Console.WriteLine("2. Metot çağrıldı"); }

    static void Metot1 (string x, string y)
    { Console.WriteLine("3. Metot çağrıldı");}

    static void Main()
    {   Metot1 ("deneme", "deneme");
        Metot1 (5, 6);
        Metot1 (10f, 56f);
    }
}
```

Metotların Aşırı Yüklenmesi

- Derleyici hangi metodu çağıracağına karar vermek için, metot bildirimi ile metot çağrımı arasındaki tam uyumu arar. Eğer tam uyumluluk bulunamaz ise parametreler arasında küçük türün büyük türe dönüşmesinin yasal olabileceği metot çağrılır.

```
static void OrnekMetot(double x, double y)
{
  Console.WriteLine("1.Metot");
}
static void OrnekMetot(byte x, byte y)
{
   Console.WriteLine("4.Metot");
}

static void Main()
{
    OrnekMetot(1919,1923);
    OrnekMetot(12, 34);
    Console.ReadLine();
}
```

Değişken Sayıda Parametre Alan Metotlar

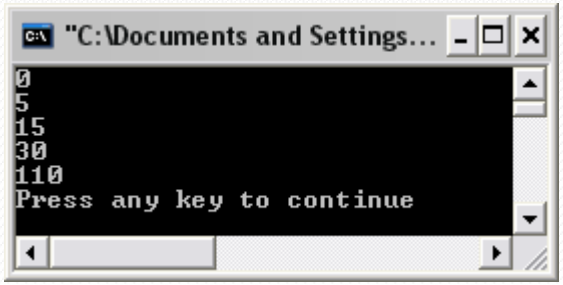
- Metotlara gönderilebilecek parametre sayısı bilinmediği zamanlarda değişik sayıda parametre alan metotlar tanımlanabilir.

- Bu metotlar parametreleri dinamik bir dizi içinde depolayarak kullanılmasını sağlarlar.

- Değişken sayıda parametre alan metot tanımlamak için **params** anahtar sözcüğü kullanılır .

Değişken Sayıda Parametre Alan Metotlar

```
using System;
class paramsa
{
    static int Toplam(params int[] sayılar)
    {
        if (sayılar.Length == 0)
            return 0;
        int toplam = 0;
        foreach (int sayi in sayılar)
            toplam += sayi;
        return toplam;
    }
    static void Main()
    {
        Console.WriteLine(Toplam());
        Console.WriteLine(Toplam(5));
        Console.WriteLine(Toplam(5, 10));
        Console.WriteLine(Toplam(5, 10, 15));
        Console.WriteLine(Toplam(5, 10, 15, 30, 50));
    }
}
```



Değişken Sayıda Parametre Alan Metotlar

- Değişken sayıda parametre içeren metotlar, aşırı yüklenmiş metotlar olduğunda değerlendirmeye alınmazlar (params ifadesi). Bu programdaki gibi metodumuzda değişken parametre yanında bir ya da daha fazla normal sabit parametre de olabilir. Ancak değişken parametre mutlaka en sonda yazılmalıdır.

```
using System;
class Metotlar
{
    static void Metot1(int x, int y)
    { Console.WriteLine("1. metot çağrıldı."); }
    static void Metot1(int x, params int[] y)
    { Console.WriteLine("2. metot çağrıldı."); }
    static void Main()
    { Metot1(3, 6); }
}
                                        // 1. metot çağrılır
```

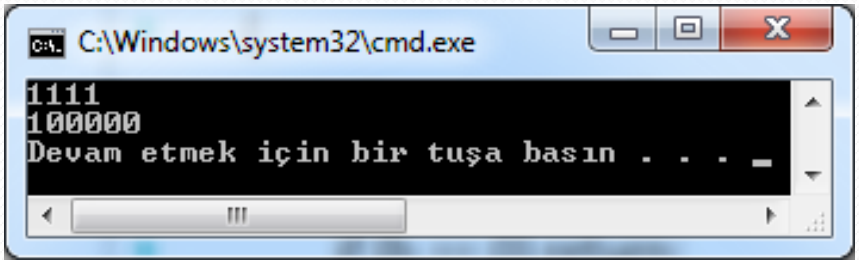
Değişken Sayıda Parametre Alan Metotlar

```
using System;
class Metotlar
{
    static void Metot1(int x, int y)
    { Console.WriteLine("1. metot çağrıldı."); }
    static void Metot1(int x, params int[] y)
    { Console.WriteLine("2. metot çağrıldı."); }
    static void Main()
    { Metot1(3, 6, 8); }
}
                              //Burada 2. metot çağrılır.
```

Rekürsif metotlar

- Kendi kendini çağıran metotlara özyineli(recursive) metot denilir. Bu metotlarda, çağrımı sonlandıran bir kontrol yapısı da olmalıdır.
- **Örnek:** 10 luk sayıyı 2 lik sayıya dönüştüren program

```
using System;
class Program
{
    static void BitYaz(int b)
    {   if (b == 0) return;
            BitYaz(b >> 1);
            Console.Write(b & 1);
    }
    static void Main()
    {   BitYaz(15); Console.WriteLine();
        BitYaz(32); Console.WriteLine();
    }
}
```



Main Metodu

- Main metodunu diğer metotlardan temel olarak farkı yoktur, tek farkı programın çalışmaya başladığı nokta olmasıdır.

- Main metodunun int türünden bir geri dönüş değeri de olabilir. Bu değer işletim sistemine programın çalışması hakkında bilgi vermek amacıyla kullanılır. Örneğin Main metodu 0 değerini döndürürse işletim sistemi programın düzgün şekilde sonlandırıldığını, 1 değerini döndürürse de hatalı sonlandırıldığını anlayabilecektir.

- Main metodu parametre de alabilir. Komut satırından çalıştırıldığında verilen parametreleri bir **string** dizisi içinde alarak işleyebilir.

- static int Main (**string [] args**)

Main Metodu

- static int Main (**string [] args**)
- Burada komut satırından girilen argümanlar string[] türündeki args dizisine aktarılıyor. Bu diziyi programımızda gönlümüzce kullanabiliriz. Örneğin programımız deneme.exe olsun. Komut satırından;
- ***deneme ayşe bekir rabia***
- girilirse ilk sözcük olan deneme ile programımız çalıştırılır ve ayşe, bekir ve rabia sözcükleri de string[] türündeki args dizisine aktarılır. komut satırında program adından sonra girilen ilk sözcük args dizisinin 0. indeksine, ikinci sözcük 1. indeksine, üçüncü sözcük 2. indeksine vs. aktarılır. Örnek bir program:

```
using System;
class Metotlar
 { static void Main(string[] args) {
 Console.WriteLine("Komut satırından şunları girdiniz: ");
 foreach(string i in args) Console.WriteLine(i); }
 }
```

System.Math Sınıfı()

- .Net sınıf kütüphanesinde belirli matematiksel işlemleri yapan birçok metot ve iki tane de özellik vardır. System.Math sınıfındaki metotlar static oldukları için bu metotları kullanabilmek için içinde bulundukları sınıf türünden bir nesne yaratmaya gerek yoktur. System.Math sınıfındaki iki özellik matematikteki pi ve e sayılarıdır. PI ve E özellikleri double türünden değer tutarlar.

```
using System;
class Metotlar
{
   static void Main()
    {
       double e=Math.E;
       double pi=Math.PI;
       Console.Write("e->"+e+" pi->"+pi);
     }
 }
```

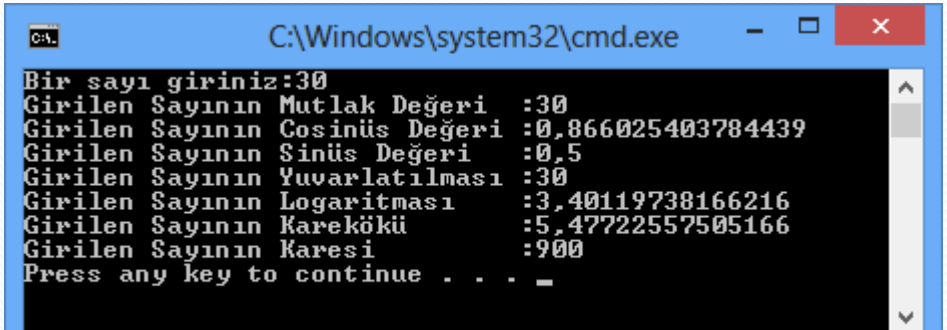
System.Math Sınıfı()

- System.Math sınıfındaki bütün metotlar

Abs(x)	Bir sayının mutlak değerini tutar.
Cos(x)	Sayının kosinüsünü tutar.
Sin(x)	Sayının sinüsünü tutar.
Tan(x)	Sayının tanjantını tutar.
Ceiling(x)	x sayısını x ten büyük en küçük tamsayıya yuvarlar.
Flor(x)	x sayısını x ten küçük en büyük tamsayıya yuvarlar.
Max(x,y)	x ve y sayılarından en büyüğünü bulur.
Min(x,y)	x ve y sayılarından en küçüğünü bulur.
Pow(x,y)	x üzeri y yi hesaplar.
Round(x)	x i ne yakın tam sayıya yuvarlar.
Sqrt(x)	x in karekökünü bulur.
Log(x)	x sayısının e tabanında logaritmasını alır.
Exp(x)	e üzeri x'i hesaplar.
Log10(x)	x sayısının 10 tabanında logaritmasını hesaplar.

System.Math Sınıfı()

```
using System;
class argümanliste
{
  static void Main()
  {
     double a;
     Console.Write("Bir sayı giriniz:");
     a = Convert.ToDouble(Console.ReadLine());
     Console.WriteLine("Girilen Sayının Mutlak Değeri  :" + Math.Abs(a));
     Console.WriteLine("Girilen Sayının Cosinüs Değeri :" + Math.Cos(a*Math.PI /180));
     Console.WriteLine("Girilen Sayının Sinüs Değeri   :" + Math.Sin(a*Math.PI /180));
     Console.WriteLine("Girilen Sayının Yuvarlatılması :" + Math.Round(a));
     Console.WriteLine("Girilen Sayının Logaritması    :" + Math.Log(a));
     Console.WriteLine("Girilen Sayının Karekökü       :" + Math.Sqrt(a));
     Console.WriteLine("Girilen Sayının Karesi         :" + Math.Pow(a, 2));
  }
}
```



Sınıflar(Classes)

- Nesne Yönelimli Programlama(NYP) nedir?
- Çözülmesi istenen problemi çeşitli parçalara ayırıp her bir parça arasındaki ilişkiyi gerçeğine uygun şekilde belirleme tekniğine NYP denir. Nesnelerin kaynak kodundaki karşılığı sınıflar dır.

- NYP de her şeyin nesnelere dayalı olması gerekir. C# 'ta her şey nesne olduğu için %100 nesne yönelimlidir.

Sınıflar(Classes)

- Sınıflar nesne yönelimli programlamanın en önemli ögesidir. Sınıflar sayesinde programlarımızı parçalara bölüp karmaşıklığını azaltırız.
- Sınıf bildirimleri **class** anahtar sözcüğü kullanılarak yapılır ve sınıf bildirimleri için bellekte yer tahsis edilmez.
- Sınıfların üye elemanları değişkenler(özellik) ve metotlardır.
  - Metotlar bir veya daha fazla komutun bir araya getirilmiş halidir; parametre alabilirler, geriye değer döndürebilirler.
  - Özellikler ise bellek gözeneklerinin programlamadaki karşılıklarıdır. Bu bakımdan özellikler değişkenlere benzerler. Aradaki en temel fark değişkenlerin bir metot içinde tanımlanıp yalnızca tanımlandığı metot içinde etkinlik gösterebilmesine rağmen özelliklerin tıpkı metotlar gibi bir üye eleman olmasıdır. Bu bakımdan özelliklerin tuttuğu değerlere daha fazla yerden erişilebilir.

Sınıflar(Classes)

- Her metot ve özellik için, bildirim yapmadan önce erişim belirleyici türünü bildirilmesi gerekir. Erişim belirleyiciler bir metoda ve özelliğe nereden erişileceğini tanımlar.

```
class sınıf_adı
{
        <erişim belirleyici> <veri türü> özellik1;
        <erişim belirleyici> <veri türü> özellik2;

        <erişim belirleyici> <geri dönüş tipi> metot1(parametreler)
        {
              metot ifadeleri, deyimleri
        }

        <erişim belirleyici> <geri dönüş tipi> metot2(parametreler)
        {
              metot ifadeleri, deyimleri
        }
}
```

Sınıflar(Classes)

- C# dilinde 5 tane erişim belirleyici vardır.
- **public:** erişim belirteci ile tanımlanan sınıfın bir özelliğine ya da metoduna istenilen yerden erişime izin verilmiş olur. public üye elemanlar genelde sınıfın dışa sunduğu ara yüzü temsil eder.
- **private:**erişim belirteci ile tanımlanan üyelere ancak tanımlandıkları sınıfın içindeki diğer üye elemanlar erişebilir, dışarıdan erişmek mümkün değildir.
- **protected:**erişim belirteci sınıf türemesi yoksa private ile aynıdır. Fakat bir sınıftan başka bir sınıf türetiliyorsa private üyeler diğer sınıfa aktarılmaz, sadece public ve protected elemanlar aktarılırlar.

Sınıflar(Classes)

- Sadece sınıflar için kullanılan diğer erişim belirleyicileri

- **internal:** internal olarak tanımlanan öğe, bulunduğu assembly'nin (Dll veya Exe dosyası) içinde erişilebilirdir. Dll veya Exe dosyasının içerisinde erişim için kısıtlama yoktur, ama dışarıdan erişilemez. (Her hazırlanan proje farklı assembly dosyası oluşturur)

- **protected internal:** protected internal erişim belirleyicisi, protected ve internal erişim belirleyicilerinin VEYA (OR) işlemiyle birleştirilmiş halidir. Tanımlandığı class'ın içinde ve o class'tan türetilmiş diğer class'ların içinde erişilebilir. Ayrıca, aynı assembly içinde olmasalar dahi, tanımlandığı class'tan türetilmiş diğer class'ların içinde de erişilebilirdir.

Sınıf Bildirimi

- Sınıf bir veri yapısıdır. Gerçek hayattaki herhangi bir nesne bu yapı ile modellenebilir.
- Sınıf tanımlamaları yapılırken amaca uygun isimler seçilmelidir.
- Örneğin bir kredi kartı hesabı için bir çok özellik vardır, hesap no,limit,kart sahibi. Bu bizim için gerçek hayattaki bir modeli temsil eden bir yapıdır. Örnekte sadece özellikleri olan bir sınıf tanımlanmıştır. Elemanlarının hepsi public tanımlandıkları için sınıfın tüm üye değişkenlerine dışarıdan erişilebilir. Erişim belirteci verilmeyen tanımlamaların varsayılan değeri **private** olur.

```
class KrediHesabi
{
    public int HesapNo;
    public double Limit;
    public string KartSahibi;
}
```

Sınıf Nesneleri Tanımlama

- Oluşturulan sınıflardan nesneler tanımlamak için **new** anahtar sözcüğü kullanılır.

- Nesnelerin bildirilmesi ve tanımlanması şu şekilde olur:
  - Ogrenci ogr1; (Bildirim)

- Bildirim ile nesneye erişim için bir değişken tanımlanmış olur. Fakat nesne bellekte oluşturulmamıştır. Yer tahsisatı yapabilmek için **new** kullanılır. Bunun için de;
  - Ogrenci ogr1 = new Ogrenci();

  kullanılır.

- ogrenci ogr1; ***// nesneyi gösteren referans tanımlandı***
- ogr1 = new Ogrenci(); ***// Nesne için bellek tahsil edildi***

Sınıf Nesneleri Tanımlama

- Aynı proje dosyası içersinde birden fazla sınıf tanımlamak mümkündür. Bu işlem karmaşıklığı azaltmak için farklı dosyalarda da yapılabilir. Derleyici bu dosyaların hepsini birden derleyebilir.

- Örnek derlendiğinde bazı uyarı mesajları çıkacaktır. Bu mesajlar ileride anlatılacak özel metotlarla giderilebilir.

- Sınıflar, new anahtar sözcüğü ile tanımlandığında sınıfın içerisindeki bütün üyeler varsayılan değere atanır.

- Sınıftan tanımlanan bir nesnenin üyelerine '.' operatörü ile ulaşılır. Public olarak tanımlanmış üye elemanların değerleri değiştirilebilir.

Sınıf Bildirimi

- **Örnek:**
- **using** System;
- **class** SinifIsmi
- { **public int** ozellik1=55; // başlangıç değerleri verildi.
- **private string** ozellik2; // nesne oluşturulduğu anda varsayılan değerler atanıyor.
- **float** ozellik3; // private yazılmasa bile private gibi davranır
- **public int** metot1(**int** a,**int** b)
- { **return** a+b; }
- **private void** metot2(**string** a)
- { Console.WriteLine(a); }
- }
- **class** EsasSinif {
- **static void** Main() {//private erişim olmaz
- SinifIsmi nesne=new SinifIsmi(); Console.WriteLine(nesne.ozellik1); Console.WriteLine(nesne.ozellik2); Console.WriteLine(nesne.ozellik3);
- Console.WriteLine(nesne.metot1(2,5)); nesne.metot2("deneme"); } }

Birden Fazla Sınıf Nesnesi Tanımlama

- Bir sınıftan birden fazla nesne de tanımlanabilir ve bu nesnelerle sınıfın özellik ve metotlarına erişilebilir. Fakat bu tanımlanmış nesneler ile işlem yapılırken referans tiplerde anlatılan kurallara dikkat edilmelidir. Değer tipleri birbirine atanırken bitsel olarak kopyalanır. Referans türleri böyle değildir.

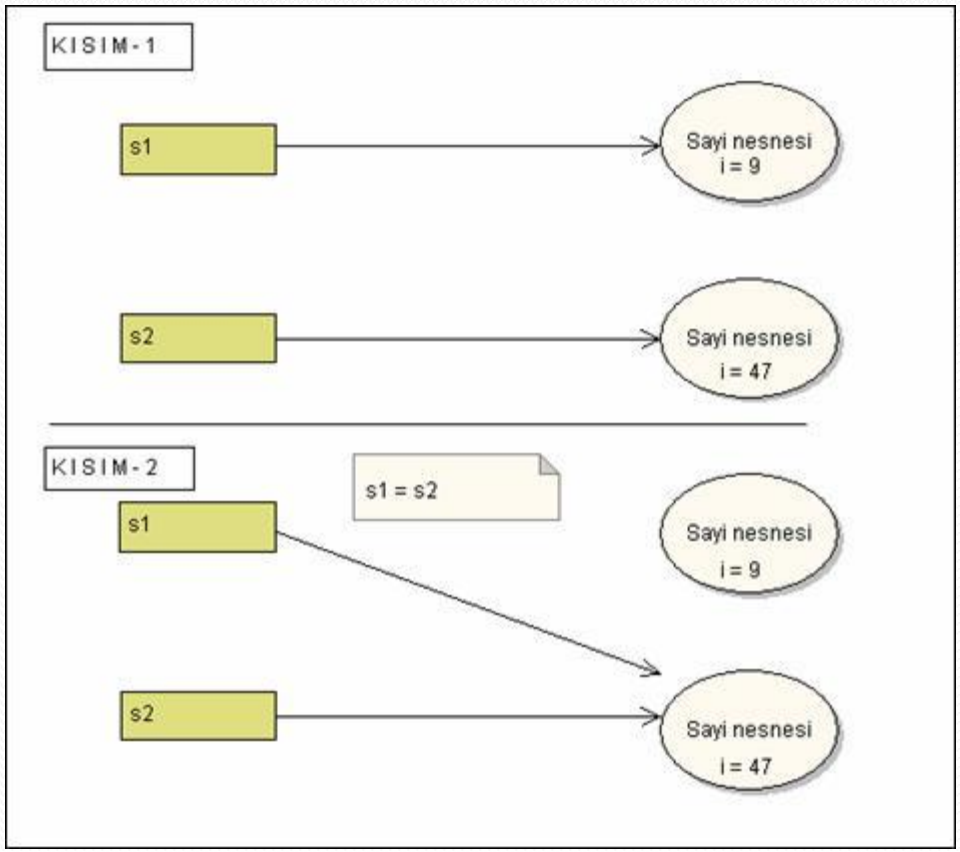
- Tanımlanmış iki nesne değeri atama operatörü ile birbirine eşitlenirse her iki nesne de aynı bellek bölgesindeki verileri temsil eder. Birinde yapılan değişikler diğerini de etkiler.

Birden Fazla Sınıf Nesnesi Tanımlama

```
using System;
class Sayi {
            public int i; }

class NesnelerdeAtama {
  public static void Main(string [] args) {
            Sayi s1 = new Sayi();
            Sayi s2 = new Sayi();

    s1.i = 9;     s2.i = 47;
    Console.WriteLine("1: s1.i: " + s1.i +", s2.i: " + s2.i);
    s1 = s2; // referanslar kopyalaniyor.. nesneler degil
    Console.WriteLine("2: s1.i: " + s1.i +", s2.i: " + s2.i);
    s1.i = 27;
    Console.WriteLine("3: s1.i: " + s1.i +", s2.i: " + s2.i);
  }
}
```

Sonuçları yorumlayınız
1: s1.i: 9, s2.i: 47
2: s1.i: 47, s2.i: 47
3: s1.i: 27, s2.i: 27

Birden Fazla Sınıf Nesnesi Tanımlama



Örnek

Bu program bir metoda girilen iki parametreyi ve bunların çarpımını ekrana yazar. Eğer bir dikdörtgen söz konusu olduğunu düşünürsek dikdörtgenin enini, boyunu ve alanını ekrana yazar.

```
using System;
class Dortgen {
public int En;
public int Boy;
public int Alan() {
int Alan=En*Boy; return Alan; }
public void EnBoyBelirle (int en,int boy) {
En=en; Boy=boy; }
public void Yaz() {
Console.WriteLine("****************");
Console.WriteLine("En:{0,5}",En);
Console.WriteLine("Boy:{0,5}",Boy);
Console.WriteLine("Alan:{0,5}",Alan());
Console.WriteLine("****************"); } }
```

```
class AnaSinif
{ static void Main()
{ Dortgen d1=new Dortgen();
d1.EnBoyBelirle(20,50);
d1.Yaz();
Dortgen d2=new Dortgen();
d2.EnBoyBelirle(25,12); d2.Yaz();
}
 }
```

Örnek

Programdaki hatanın nedenini bulunuz

```
using System;
class Dortgen {
public int En=20; public int Boy=5;
public int Alan()
{ int Alan=En*Boy; return Alan; }
static void Main()
{Console.WriteLine("****************")
; Console.WriteLine("En:{0,5}",En);
Console.WriteLine("Boy:{0,5}",Boy);
Console.WriteLine("Alan:{0,5}",Alan());
Console.WriteLine("****************");
 } }
```

Örnek

Programdaki hatanın nedenini bulunuz

```
using System;
class Dortgen {
public int En=20; public int Boy=5;
public int Alan()
{ int Alan=En*Boy; return Alan; }

static void Main()
{Console.WriteLine("****************")
; Console.WriteLine("En:{0,5}",En);
Console.WriteLine("Boy:{0,5}",Boy);
Console.WriteLine("Alan:{0,5}",Alan());
Console.WriteLine("****************");
 } }
```

- Cevap :
- Burada En ve Boy özellikleriyle Alan() metodu tek başına kullanılamaz.
- Önce Dortgen sınıfı türünden bir nesne yaratıp bu nesne üzerinden bu özellik ve metotlara erişilmelidir.
- Eğer özellikler ve metod static olsaydı direk kullanılablirdi.

Sınıf içinde başka bir sınıf tanımlama

Sınıflar ayrı ayrı tanımlanabildiği gibi sınıf içinde başka bir sınıfta oluşturulabilir. İçteki sınıfın eleman veya metotlarına erişmek için kurallar aynıdır.

```
using System;
class Sinif
{   int x=10;     static int y = 0;
    class Sinif2
    {   public int x = 0;          static public int y=10;
    }
    static void Main()
    {   Sinif a = new Sinif();
        Sinif2 b = new Sinif2();
        a.x = 7; Sinif.y = 15;
        b.x = 12; Sinif2.y = 1;
    }
}
```

Lab-Uygulama

- Evinizin üyelerinin (annemizin, babamızın vs.) özelliklerini gireceğiniz, yaşlarını hesaplayacağınız, mesleklerini belirleyeceğiniz Evim sınıfını oluşturunuz.
- Bu sınıfa ait özellikleri ve metotları belirleyiniz.

this Anahtar Sözcüğü

- Metotlar içinde nesnelerin özellikleri ve diğer metotlar kullanılabilir. Aslında burada nesnenin referansı da gizli bir şekilde aktarılmaktadır.

- Birçok durumda sorun yaşanmamakla birlikte bazen nesnenin üyeleri ile metot içindeki tanımlamalar aynı olabilir. Bu durumda metot değişkenleri ile nesnenin üyelerini birbirinden ayırmak için **this** anahtar sözcüğü kullanılır.

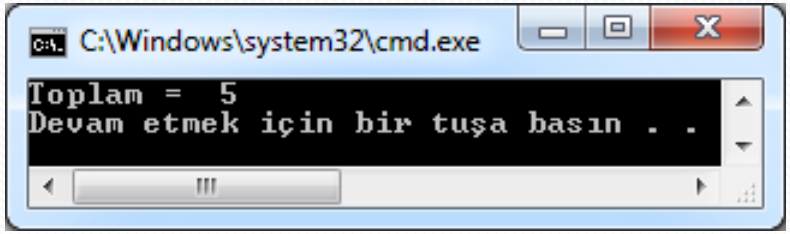
this Anahtar Sözcüğü

- this anahtar sözcüğü ilgili nesnenin referansını belirtir.
- **this** anahtar sözcüğü, içinde bulunulan nesneye ait bir referans döndürür
- Bunun sayesinde nesnelere ait global alanlara erişme fırsatı bulunur;

this Anahtar Sözcüğü

```
using System;

class Program
   {
   int topla; // Global alandaki değişken
   public void sayiTopla(int topla) // yerel alandaki değişken
   {
      this.topla += topla;
   }
    public void sayiyiEkranaYaz() {
          Console.WriteLine("Toplam =  " + topla);
 }
      static void Main(string[] args)
      {  Program th = new Program();
         th.sayiTopla(2); th.sayiTopla(3); th.sayiyiEkranaYaz();
      }
   }
```

This olmasaydı sonuç 0 olurdu. Neden?



set ve get Anahtar Sözcükleri

- Bir nesnenin üye değişkenlerine değer atarken ya da içerisindeki değeri kullanırken belli kontrollerin yapılması gerekiyorsa ya da kod bloklarının çalıştırılması isteniyorsa **set** ve **get** ifadeleri kullanılır.

- **set** sözcüğü nesnenin özelliklerine değer atandığında çalışır.
- **get** sözcüğü ise özellik değeri okunduğunda ya da farklı bir ifadeye aktarılmaya çalışıldığında çalışır.

- set ve get yönteminin daha iyi anlaşılabilmesi için aynı işlemi yapan iki örneği ; 1.örnekte metotlar kullanılarak, 2. örnekte ise set ve get yöntemi kullanarak yapalım

set ve get Anahtar Sözcükleri

1.Örnek

```
using System;
class MetotSinif
{  int Sayi;
   public void SayiBelirle(int sayi)
   {    if(sayi<0)      Sayi=0;
        else      Sayi=sayi;
   }
   public int SayiyiAl()
   {  if(Sayi>100)
        return Sayi/100;
      else
        return Sayi;
   }
}
```

```
class AnaSinif
{  static void Main()
   {
   MetotSinif nesne=new MetotSinif();
      nesne.SayiBelirle(34);
      Console.WriteLine(nesne.SayiAl());
   }
}
```

set ve get Anahtar Sözcükleri 2.Örnek

```
using System;
class Set_GetSinif
{  int Sayi;
   public int SahteOzellik    {
      set
      {  if(value<0)  Sayi=0;
         else            Sayi=value;
      }
      get
      {  if(Sayi>100)   return Sayi/100;
         else     return Sayi;
      }
   }
}
```

```
class AnaSinif
 {  static void Main()
    {
Set_GetSinif nesne=new Set_GetSinif();
nesne.SahteOzellik=110;
Console.WriteLine(nesne.SahteOzellik);
}
 }
```

set ve get Anahtar Sözcükleri

- 2.Örnekte SahteOzellik adlı bir özellik oluşturduk. Bu özellik gerçekten de sahtedir. Özelliğe bir değer atanmaya çalışıldığında set bloğundaki, özellik kullanılmaya çalışıldığında da get bloğundaki komutlar çalıştırılır. Aslında C# kütüphanesindeki özelliklerin çoğu bu yöntemle oluşturulmuştur.
  - Örneğin ileride göreceğimiz Windows programlamada bir buton nesnesinin Text özelliğini değiştirdiğimizde butonun üzerindeki yazı değişir. Halbuki klasik özelliklere değer atama yönteminde sadece özelliğin değeri değişirdi. Bu yöntemde ise bir özelliğe değer atadığımızda çalışacak komutlar yazabiliyoruz.
- Programdaki value sözcüğü özelliğe girilen değeri tutar. Özelliğe girilen değer hangi türdeyse o türde tutar. Ayrıca bu oluşturulan SahteOzellik özelliğinin metotlara oldukça benzerdir. Görüldüğü gibi, değişkenin değerini değiştirdiğimizde çalışmasını istediğimiz kodları set blokları arasına yazarız.

set ve get Anahtar Sözcükleri

- Herhangi bir sahte özelliğin set veya get bloklarından yalnızca birini yazarak o özelliği salt okunur veya salt yazılır hâle getirebiliriz. Örneğin Array sınıfının Length özelliği salt okunur bir özelliktir.

- **NOT:** C# 2.0'da ya da daha üst versiyonlarda bir sahte özelliğin set ve get bloklarını ayrı ayrı private veya public anahtar sözcükleriyle belirtebiliyoruz. Yani bir özelliğin bulunduğu sınıfın dışında salt okunur ya da salt yazılır olmasını sağlayabiliyoruz.

Özelliklerde Erişim Belirleyiciler

```
private int m_uzunluk;
public int Uzunluk
{
   get
   {return m_uzunluk;
   }

   set
   { m_uzunluk = value;
   }
}
```

get ve set blokları public olarak tanımlıdır.

```
private int m_uzunluk;
public int Uzunluk
{
   get
   {return m_uzunluk;
   }

   private set
   { m_uzunluk = value;
   }
}
```

get ve set blokları; C#2.0'dan itibaren farklı erişim belirleyicileri ile belirlenebilmektedir.

- Değişken için bildirilen erişim belirleyicisi; get veya set bloğunda tanımlanan erişim belirleyicisinden daha yüksek erişim yetkisine sahip olmalıdır.

set ve get Anahtar Sözcükleri

- **NOT:** get ve set anahtar sözcükleriyle erişim belirleyiciler kullanırken uymamız gereken bazı özellikler vardır:
- Daima özellik bildiriminde kullanılan erişim belirleyicisi get veya set satırında kullanılan erişim belirleyicisinden daha yüksek seviyeli olmalıdır. Örneğin özellik bildiriminde kullanılan erişim belirleyici private ise get veya set satırında kullanılan erişim belirleyici public olamaz.
- get veya set satırında kullanılan erişim belirleyici özellik bildiriminde kullanılan erişim belirleyiciyle aynı olmamalıdır. (Zaten gereksizdir.)
- Yani işin özü set ve get satırlarındaki erişim belirleyicileri yalnızca, özelliği public olarak belirtmiş ancak özelliğin set veya get bloklarının herhangi birisini private yapmak istediğimizde kullanılabilir.
- get veya set için erişim belirleyicisi kullanacaksak sahte özelliğin bloğu içinde hem get hem de set bloğunun olması gerekir.

Dizililerin metot ve özelliklerde kullanılması

- Aslında tanımladığımız her dizi Array sınıfı türünden bir nesnedir. İşte bu yüzden tanımladığımız dizilerle Array sınıfının metot ve özelliklerine erişebiliriz.

- Örneğin Length Array sınıfı türünden bir özelliktir ve dizinin eleman sayısını verir. DiziAdi.Length yazılarak bu özelliğe erişilebilir.

- Metot ve özelliklerin geri dönüş tipi bir dizi olabilir. Örnek:

Dizililerin metot ve özelliklerde kullanılması

```
using System;
class YardimciSinif
{   public int[] Dizi={7,4,3};
    public int[] Metot()
    {  int[] a={23,45,67};
       return a;
    }
}
class AnaSinif
{  static void Main()
    {    YardimciSinif nesne=new YardimciSinif();
       Console.WriteLine(nesne.Dizi[0]);
       Console.WriteLine(nesne.Metot()[2]);
    }
}
```

Dizililerin metot ve özelliklerde kullanılması

- Bir dizinin türünü istenilen türe dönüştürmeye yarayan bir sınıf örneği bulunmamaktadır.

- C#'ta diziler arasında bilinçsiz tür dönüşümü mümkün olmadığı gibi, bu işi yapacak metot da yok.

- Aşağıda verilen örnekteki metotlar bu iş için kullanabilirsiniz.

- **HATIRLATMA:** Metotların parametresindeki dizi, Array DiziAdi yöntemiyle oluşturulduğu için bu dizinin elemanlarına klasik indeksleme yöntemiyle erişemeyiz. Bu yüzden bu dizinin elemanlarına ulaşmak için Array sınıfının GetValue() metodu kullanıldı.

Dizililerin metot ve özelliklerde kullanılması

```
using System;
class Donustur
{   public static int[] Inte(Array dizi)
  {
    int[] gecici=new int[dizi.Length];
    for(int i=0;i<dizi.Length;i++)

gecici[i]=Convert.ToInt32(dizi.GetValue(i));
    return gecici;
  }
  public static string[] Stringe(Array dizi)
  {
    string[] gecici=new string[dizi.Length];
    for(int i=0;i<dizi.Length;i++)
      gecici[i]=dizi.GetValue(i).ToString();
    return gecici;
  }
}
```

```
class AnaProgram
{
  static void Main()
  {
    string[] a={"2","5","7","9"};
    int[] b=Donustur.Inte(a);
    Console.WriteLine(b[1]+b[3]);
    int[] c={2,7,9,4};
    string[] d=Donustur.Stringe(c);
    Console.WriteLine(d[0]+d[3]);
  }
}
//farklı türleri kendiniz ekleyebilirsiniz.
```

Yapıcı Metotlar(Constructors)

- Bir nesnenin dinamik olarak yaratıldığı anda otomatik olarak çalıştırılan metotlar vardır. Bu metotlar sayesinde bir nesnenin üye elemanlarına ilk değerler verilebilir ya da gerekli ilk düzenlemeler yapılabilir. Bu metotlara yapıcı metotlar (constructors) denir.

- Yapıcı metot tanımlanmasa dahi her sınıfın varsayılan bir yapıcı metodu (default constructor) mutlaka bulunur.

- **Daha önceden bahsettiğimiz gibi bir nesne oluşturulduğunda sayısal değerler için 0, referanslar için null, bool türü için false atanır, bu atama bir yapıcı metot ile gerçekleştirilir.**

Yapıcı Metotlar(Constructors)

```
using System;
 class Deneme
 {
Deneme() { Console.WriteLine("Bu sınıf türünden bir nesne yaratıldı."); }
 }
 class AnaProgram
 {
    static void Main()
    {
       Deneme a=new Deneme();
    }
 }
```

Varsayılan Yapıcı Metot (Default Constructor)

- **Genel Özellikleri:**
- **Yapıcıların dönüş değeri yoktur, isimleri sınıf ile aynı isimdir. Varsayılan yapıcı metodun parametresi yoktur. Nesnenin üye elemanlarına varsayılan değerlerini atar.**

- **Yapıcı metotlar bir değer tutamaz ve normal metotlardan farklı olarak void anahtar sözcüğü de kullanılmaz.**

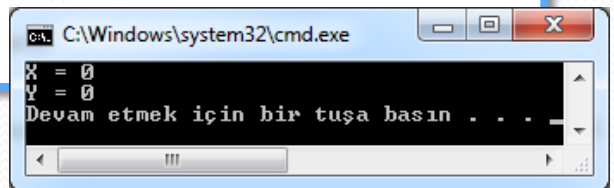
- **Yapıcı metotlar dışarıdan çağrıldığı için genelde public olarak tanımlanırlar. private olursa dışarıdan erişilemez**

- **Eğer kendi yapıcı metodumuzu tanımlarsak varsayılan yapıcı metotlar çalıştırılmaz.**

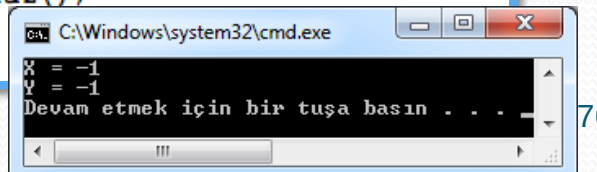
- **Yapıcı metotlar da aşırı yüklenebilir. Aşırı yükleme durumunda boş yapıcı metot mutlaka bulunmalıdır.**
- **Örneğin public Deneme();, public Deneme(int a);, public Deneme(int a, int b);**

Varsayılan Yapıcı Metot (Default Constructor)

```
using System;

class Toplama
{
    public int X;
    public int Y;

    public int Islem()
    {
        return X + Y;
    }

    public void Yaz()
    {
        Console.WriteLine("X = {0}", X);
        Console.WriteLine("Y = {0}", Y);
    }
}

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




```
using System;

class Toplama
{
    public int X;
    public int Y;

    public Toplama()
    {
        X = -1;
        Y = -1;
    }
    public int Islem()
    {
        return X + Y;
    }
    public void Yaz()
    {
        Console.WriteLine("X = {0}", X);
        Console.WriteLine("Y = {0}", Y);
    }
}
class Program
{
    static void Main()
    {
        Toplama t = new Toplama();
        t.Yaz();
    }
}
```



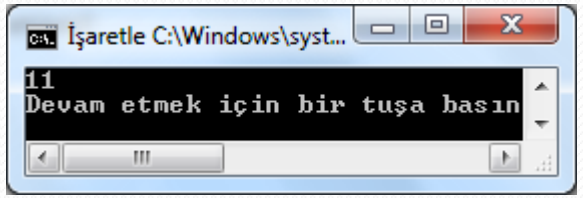
Yapıcı Metot (Default Constructor)

- Yapıcı metotlar aşırı yüklenebilir. Bu şekilde tanımlanan yapıcı metotlardan, imzası uygun olan seçilerek çalıştırılır.

```
using System;
class Deneme
{
   Deneme()   {      Console.WriteLine(0);   }
   Deneme(int a)   {      Console.WriteLine(a);   }
   Deneme(int a,int b)   {      Console.WriteLine(a+b);   }
   Deneme(int a,int b,int c)   {      Console.WriteLine(a+b+c);   }
   static void Main()
   {
      Deneme a=new Deneme(5,6);
   }
}
```

Yapıcı Metot (Default Constructor)

- Aşırı yükleme **this** anahtar sözcüğü kullanılarak da oluşturulabilir. Bu örnekte this anahtar sözcüğü sayesinde ikinci, üçüncü ve dördüncü yapıcı metotlar içeriğini aynı isimli ve üç parametre alan metottan alıyor. this anahtar sözcüğüyle kullanılan ikinci, üçüncü ve dördüncü yapıcı metotların yaptığı tek iş, birinci yapıcı metoda belirli parametreleri göndermek oluyor.
- **using** System;
- **class** Deneme
- { Deneme(**int** a,**int** b,**int** c) { Console.WriteLine(a+b+c); }
  Deneme():**this**(0,0,0) { }
- Deneme(**int** a):**this**(a,0,0) { }
- Deneme(**int** a,**int** b):**this**(a,b,0) { }
- **static void** Main() { Deneme a=new Deneme(5,6); }
- }



- **NOT:** this anahtar sözcüğü bu şekilde yalnızca yapıcı metotlarla kullanılabilir.

Yapıcı Metot (Default Constructor)

- Eğer bir sınıfta parametre alan bir veya daha fazla yapıcı metot varsa ve parametre almayan yapıcı metot yoksa -bu durumda varsayılan yapıcı metot oluşturulmayacağı için- Sinif nesne=new Sinif(); gibi bir satırla parametre vermeden nesne oluşturmaya çalışmak hatalıdır. Çünkü Sinif nesne=new Sinif(); satırı Sinif sınıfında parametre almayan bir yapıcı metot arar, bulamaz ve hata verir. Örnek:
- **class** A
- { // public A() {} tanımlanması gerekir yoksa hata verir
- **public** A(**int** a){}
- }
- **class** B
- { A a=new A(); }
- Derleme anında hata verir.

Kopyalayıcı Yapıcı Metot (Copy Constructors)

- Yapıcı metot parametre olarak kendi sınıfından bir nesne alıyorsa kopyalayıcı metot adı verilir.

```
using System;

class Toplama
{
    public int X;
    public int Y;

    public Toplama(int x, int y)
    {
        X = x;
        Y = y;
    }

    public Toplama(Toplama T)
    {
        X = T.X;
        Y = T.Y;
    }

    public int Islem()
    {
        return X + Y;
    }

    public void Yaz()
    {
        Console.WriteLine("X = {0}", X);
        Console.WriteLine("Y = {0}", Y);
    }
}
```

```
class Program
{
    static void Main()
    {
        Toplama t = new Toplama(15, 22);
        t.Yaz();

        Toplama t2 = new Toplama(t);
        t2.Yaz();
    }
}
```

Yıkıcı (Destructors) Metotlar

- Nesnelere erişim mümkün olmadığı durumlarda bu nesnelerin heap'ten silinmesi gerekir, çünkü bu nesneler bellek bölgesi işgaline sebep olur. C++' ta bu işlemi kullanıcı kendi yapmak zorundadır.

- C#' ta bu işlem için Gereksiz Nesne Toplayıcısı (Garbage Collector) mekanizması mevcuttur. Bu mekanizma gereksiz nesnelerin tuttuğu referans bölgelerini iade eder, ancak bu nesnelerin faaliyet alanlarının ne zaman iade edileceği kesin olarak bilinmez.

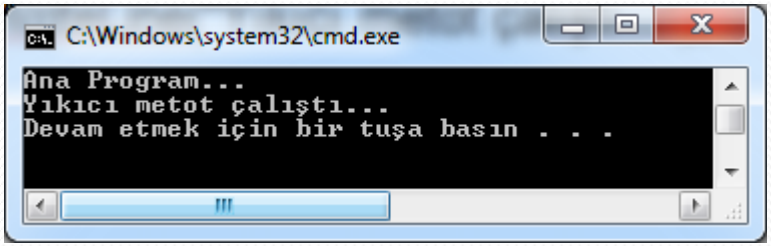
- Bu durumda yıkıcı metotlar (destructors) bellek iade işleminden hemen önce çalışarak bu belirsizliğin çözümünde yardımcı olurlar.

Yıkıcı (Destructors) Metotlar

- Yıkıcı metotlar bildirilirken sınıf isminin önüne “~” (tilda) işareti eklenir.
- Herhangi bir dönüş değeri ve parametresi yoktur.
- Erişim belirteci (public, private) de kullanılmaz.
- Bir sınıfın sadece bir tane yıkıcı metodu olabilir.

Yıkıcı (Destructors) Metotlar

```
using System;
class Yikici
{
    ~Yikici()
    {
        Console.WriteLine("Yıkıcı metot çalıştı...");
    }
}
class Program
{
    static void Main()
    {
        Yikici y = new Yikici();
        Console.WriteLine("Ana Program...");

    }
}
```

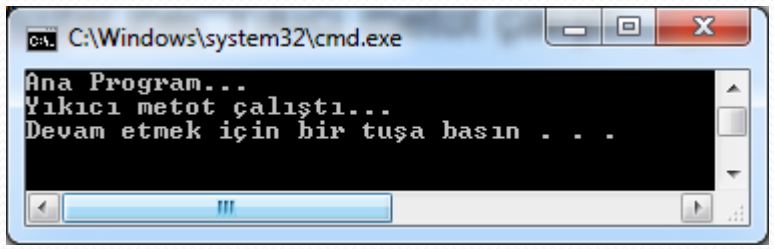


Yıkıcı (Destructors) Metotlar

- Örnek programda y nesnesi faaliyet alanını doldurmasına rağmen hemen yok edilmemiştir.
- Bir alt satırda bulunan Console. WriteLine() metodu çalışmış ardından gereksiz nesne toplayıcısı işleyerek nesneyi silmiştir.
- Nesne yok edilirken de yıkıcı metodu çalıştırılmıştır.
- Gereksiz nesne toplayıcı (Garbage Collector) programcının isteği dışında çalışmaktadır.
- System.GC sınıfı kullanılarak istenilen anda bu mekanizma çalıştırılabilir. Fakat bu tavsiye edilen bir kullanım değildir.

Yıkıcı (Destructors) Metotlar

```
using System;
class Yikici
{
    ~Yikici()
    { Console.WriteLine("Yıkıcı metot çalıştı..."); }
}
class Program
{
    static void Main()
    {
        Yikici y = new Yikici();
        GC.Collect();
        Console.WriteLine("Ana Program...");
    }
}
```



Dispose() metodu

- Oluşturulan bir nesnenin kullandığı kaynakların geri alınması için kullanılabilecek yöntemlerden bir diğeri de **Dispose()** metodudur.

- Türetmiş olduğumuz bir nesnenin hiçbir kaynak tarafından kullanılmadığı ana kadar nesneyi bellekten kaldırmaz tabi bu bazen çok uzun zaman alabilir dolayısı ile bellekte şişmeler olur ve ***OutOfMemoryException*** alırız.

- Garbage Collector'a kaynakları boşaltabilmesine yardımcı olmak için Dispose metodu devreye giriyor. Dispose metoduna ulaşmak için yazmış olduğumuz sınıfa IDisposable Interface(Arayüz)sini uygulamamız gerekmektedir.

- IDisposable Interface'si ile Dispose metodu gelir ve bu metod uygulandığı sınıftan türeyen nesnenin işi bittiği anda bellekten kaldırılmasını sağlar.
- Arayüz(Interface) kavramına daha sonra değinilecektir.

Dispose() metodu

- Öğrenci Sınıfımızı aşağıdaki kod ile oluşturalım.
-
- using System;
- namespace Samples
- { class Ogrenci: IDisposable
- { string _Adi; string _SoyAdi;
- public string Adi
- { get { return _Adi; } set { _Adi = value; }
- }
- public string SoyAdi
- { get { return _SoyAdi; } set { _SoyAdi = value; }
- }
- public DateTime DogumTarihi
- { get { return _DogumTarihi; } set { _DogumTarihi = value; }
- }
-

Dispose() metodu

- public void Dispose()
- {
- /*Dispose Metodu çağırıldığında Garbage Collector'ün SuppressFinalize metoduna this diyerek bu sınıftan türetilmiş olan nesneyi sonlandırmaya,serbest bırakmaya zorluyoruz. Bu sayede bizim bu kod ile bizzat sertbest bıraktığımız kaynaklar ile destructor'umuz tekrar uğraşmak zorunda kalmıyor buda bize performas olarak geri dönüş sağlıyor.*/
- GC.SuppressFinalize(this);
- }
- }
- }
-
- Şimdi ogrenci sınıfından bir nesne türetelim ve onu dispose metodu ile bellekten kaldıralım.
-

Dispose() metodu

- public static void Main()
- {
- Ogrenci insOgrenci = new Ogrenci();
- insOgrenci.Adi = "onur";
- insOgrenci.SoyAdi = "yılmaz";
-
- /* Bilgilerini aldığımız öğrencimizi veri tabanımıza insert ettiğimizi var sayıyoruz.ve ogrenci sınıfından türetmiş olduğumuz nesneyi dispose metodunu çağırarak bellekte yer işgal etmesini engelliyoruz.*/
- insOgrenci.Dispose();
- }
- }
- }
-

Yıkıcı (Destructors) Metotlar

- C#'da yıkıcı metotlar genelde nesnenin kullandığı bellek alanını iade etmek için kullanılmaz.

- Daha çok nesne yok edilirken bir takım işlemlerin yapılması, özellikle sınıfın global üye değişkenlerinin değerlerinin işlenmesi ya da değiştirilmesi için kullanılır.

Statik Üye Elemanlar

- Daha önce de belirtildiği gibi metotlara sınıflar üzerinden erişilir. Bazı durumlarda da ise sınıf türünden nesneler oluşturulur ve bu nesneler üzerinden ilgili metoda erişebiliriz.

- Statik elemanlar bir sınıfın global düzeydeki elemanlarıdır. Yani statik üyeleri kullanmak için herhangi bir nesne tanımlamamıza gerek yoktur.

- Bir üye elemanın statik olduğunu bildirmek için bildirimden önce **static** anahtar sözcüğü eklenir.

- Bir sınıf nesnesi oluşturulduğunda static üye elemanlar için bellekte ayrı bir yer ayrılmaz.

Statik Metotlar

- Nesne ile doğrudan iş yapmayan metotlar statik olarak tanımlanabilir.
- Statik tanımlanan metotlara **SınıfAdı.Metot** şeklinde erişilir.
- Statik metotlara nesne üzerinden erişmek mümkün değildir.

Statik Metotlar

```
using System;

class AritmetikIslem
{
    public static int Topla(params int[] dizi)
    {
        int toplam = 0;

        foreach (int eleman in dizi) toplam += eleman;

        return toplam;
    }
}
class Program
{
    static void Main()
    {
        Console.WriteLine(AritmetikIslem.Topla(1, 5, 9, 15));
    }
}
```

```
using System;

class AritmetikIslem
{
    public static int Topla(params int[] dizi)
    {
        int toplam = 0;

        foreach (int eleman in dizi) toplam += eleman;

        return toplam;
    }
}
class Program
{
    static void Main()
    {
        AritmetikIslem islem = new AritmetikIslem()
        Console.WriteLine(islem.Topla(1, 5, 9, 15));
    }
}
```

Statik Metotlar

- Main() metodu da herhangi bir nesneye ihtiyaç duymadan çalışabilmesi için static olarak tanımlanmaktadır. Eğer static olarak tanımlanmasaydı metodun çalışabilmesi için içinde bulunduğu sınıf türünden bir nesneye ihtiyaç olacaktı. Bu durumda da Main metodu işlevi ile çelişecekti. Çünkü Main metodu programımızın çalışmaya başladığı yerdir.

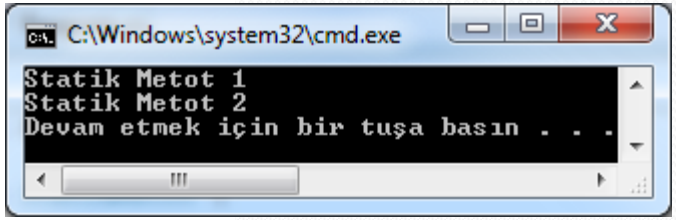
- static anahtar sözcüğü erişim belirleyiciden önce ya da sonra yazılabilir.

- static tanımlanmış metotlar içinden diğer static metotlar çağrılabilir, normal metotlar çağrılamaz. Normal metotlar nesne gerektirdiği için nesne adreslerine ihtiyaç duyarlar (bu adresler gizli şekilde ya da this ile aktarılır).

- static metotlar sınıfın global metotlarıdır ve referansları yoktur. Bu yüzden static bir metot içinden normal bir metot çağırmak geçersizdir.

Statik Metotlar

```
using System;

class StatikMetot
{
    public static void SMetot1()
    {
        Console.WriteLine("Statik Metot 1");
    }

    public static void SMetot2()
    {
        SMetot1();
        Console.WriteLine("Statik Metot 2");
    }
}
class Program
{
    static void Main()
    {
        StatikMetot.SMetot2();
    }
}
```



Statik Metotlar

```
using System;

class StatikMetot
{
    public void SMetot1()
    {
        Console.WriteLine("Statik Metot 1");
    }

    public static void SMetot2()
    {
        SMetot1();
        Console.WriteLine("Statik Metot 2");
    }
}
class Program
{
    static void Main()
    {
        StatikMetot.SMetot2();
    }
}
```

- Örneklerden görüldüğü gibi statik olmayan elemanlara ulaşabilmek için bir nesne olmalıdır. Eğer bir nesne var ise statik olmayan elemanlara erişilebilir.

Statik Değişkenler

- Herhangi bir nesne ile statik değişkenlere ulaşmamız mümkün değildir. Statik olarak tanımlanmış olan değişkenlere ancak sınıfın ismi yardımıyla ulaşılabilir.
- Statik değişkenler sınıf içersinde global özellikler tanımlamak için kullanılırlar.
- Bu değişkenler tanımlandıklarında varsayılan değere atanırlar.
- Statik üyelere ancak statik bir metot içerisinden erişilebilir.

Statik Değişkenler

```
using System;
namespace ConsoleApplication2
{
    class Statik
    {
        public static int b = 10;
        public static int deneme()
        {
            int a = 5; b = a;
            return b;
        }
    }
        class Program
        {
            static void Main(string[] args)
            {
                Statik b = new Statik();// gereksizdir ama hata vermez.
                Console.WriteLine(b.deneme());//Hata verir, Statik metotlara nesne üzerinden
                                              //erişmek mümkün değildir.
                Console.WriteLine(Statik.deneme());// deneme metoduna erişim var. 5 değerini yazar
                Console.WriteLine(Statik.b);// b değişkenine erişim var. 5 değerini yazar
                Console.WriteLine(Statik.a);// Hata verir, static veya statik olmayan metod
                                            //içerisindeki değerlere erişim yoktur.
               Console.ReadLine();
            }
        }
    }
```

Statik Yapıcı Metotlar

- Normal metotlar gibi yapıcı metotlar da static olabilirler
- Statik yapıcı metotlar bir sınıfın statik değişkenleri ile ilgili işlemler yapmada kullanılırlar.
- Bir nesne ilk defa yaratıldığında statik üye değişkenini değiştirmek için genellikle statik yapıcı metotlar kullanılır.
- Statik olmayan yapıcılarla statik değişkenleri değiştirmek mümkün değildir.
- Statik yapıcı metotlar ilk nesne tanımlandığında çalıştırılır.

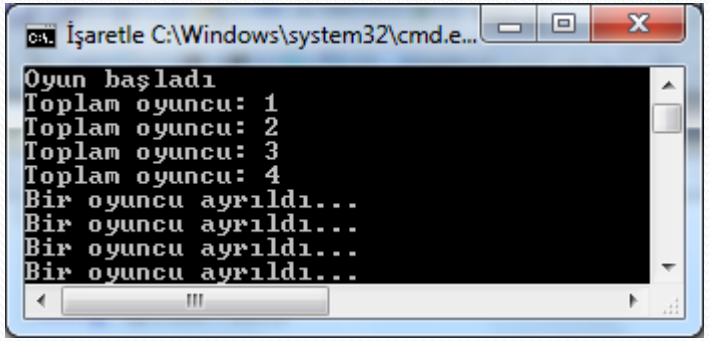
Statik Yapıcı Metotlar

```
using System;
 class Deneme
 {  static Deneme()
    {      Console.WriteLine("Static metot çağrıldı.");    }
    Deneme()
    {      Console.WriteLine("Static olmayan metot çağrıldı.");    }
    static void Main()
    {  Deneme a=new Deneme();
       Deneme b=new Deneme();
    }
 }
```

- Bu program ekrana şunları yazacaktır:
- Static metot çağrıldı.
- Static olmayan metot çağrıldı.
- Static olmayan metot çağrıldı.
- Gördüğünüz gibi bir sınıf türünden bir nesne yaratıldığında önce static metot sonra (varsa) static olmayan metot çalıştırılıyor. Sonraki nesne yaratımlarında ise static metot çalıştırılmıyor.

Statik Yapıcı Metotlar

```
using System;
class Oyuncu
{
    static int Toplam;
    Oyuncu()
    { Toplam++; Console.WriteLine("Toplam oyuncu: " + Toplam); }
    static Oyuncu()
    { Console.WriteLine("Oyun başladı"); }
    ~Oyuncu()
    {
        Console.WriteLine("Bir oyuncu ayrıldı..."); Toplam--;
    }
    static void Main()
    {
        Oyuncu ahmet = new Oyuncu();
        Oyuncu osman = new Oyuncu();
        Oyuncu ayse = new Oyuncu();
        Oyuncu mehmet = new Oyuncu();
    }
}
```



Static Sınıflar

- Bir sınıf statik olarak bildirildiğinde o sınıf türünden bir nesne yaratılamaz. Statik metotlar C# 2.0'dan itibaren dile eklenmiştir.
- Statik sınıflar içerisinde static olmayan metod veya özellik tanımlanamaz.
- Static sınıfların yapıcı metotları olmaz. Dolayısıyla yapıcı metot tanımlamaya çalışırsak derleyici hata verir.
- Daha sonra bahsedilecek olan nesne konusundaki türeme başlığında tekrar bahsedeceğimiz üzere; static sınıflar türetmeyi desteklemez.

Static Olmak, Non-Static Olmak

- **Bir sınıfa ait her nesne için özel bir iş söz konusu ise, üye elamanları veya metot non-static tanımlanmalıdır, Eğer yapılan iş genel bir iş ise, üye elamanları veya metot static olarak tanımlanmalıdır.**
- **Örnek:**

```
class Urun
    {
        public int Id;
        public string Ad;
        public double Fiyat;
        public double KdvOran;
    }
```

Static Olmak, Non-Static Olmak

```
class Program
    {
        static void Main(string[] args)
        {
            Urun urun1 = new Urun();
            urun1.Id = 1;
            urun1.Ad = "Monitör";
            urun1.Fiyat = 200;
            urun1.KdvOran = 0.18;
            Urun urun2 = new Urun();
            urun2.Id = 2;
            urun2.Ad = "Klavye";
            urun2.Fiyat = 80;
            urun2.KdvOran = 0.18;
        }
    }
```

KdvOran her iki nesne içinde ortak olduğuna göre yeniden tanımlamaya gerek yoktur. Bu yüzden static tanımlanması hafıza ve hız açısından daha önemlidir.

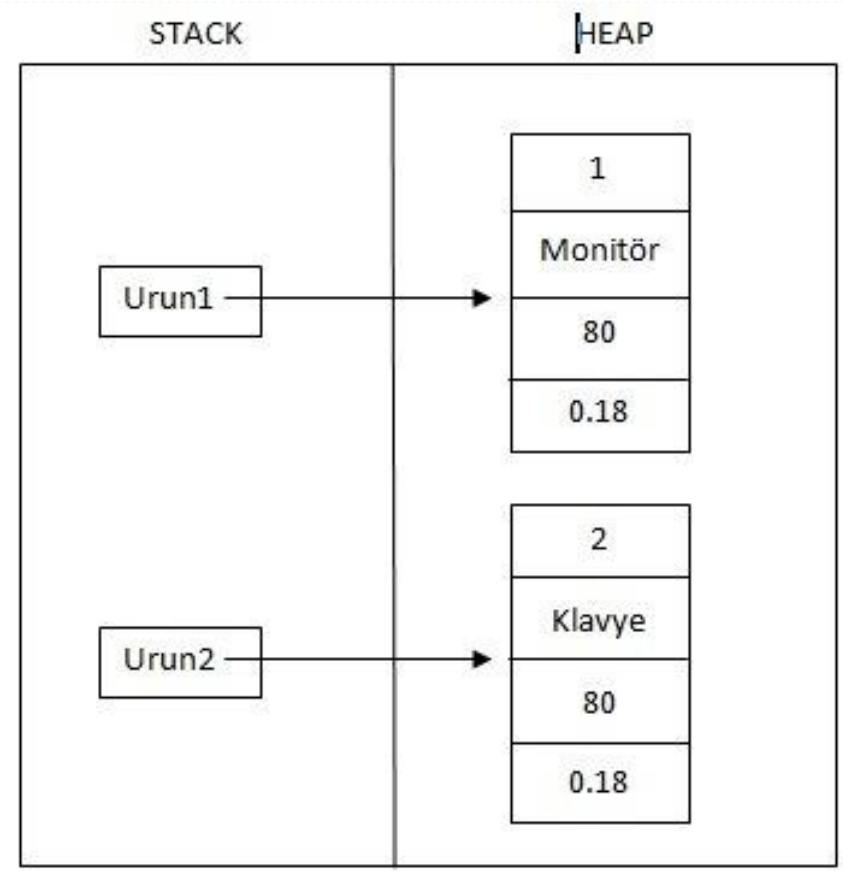
Static Olmak, Non-Static Olmak

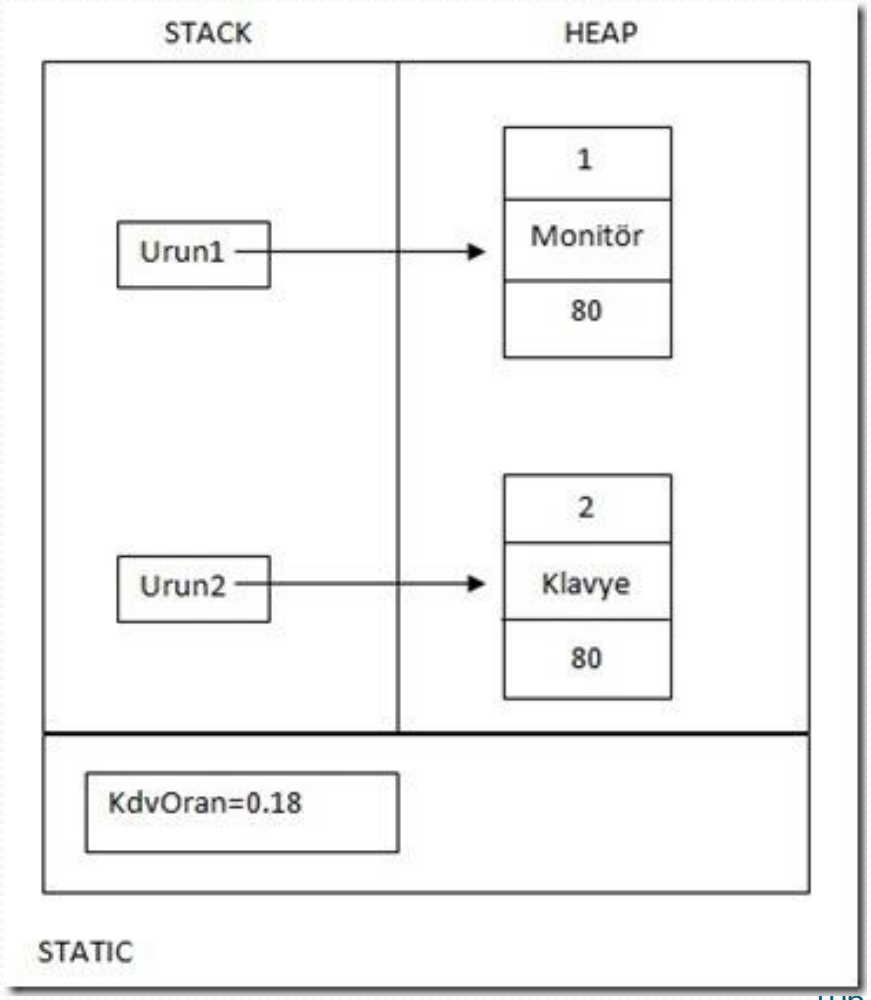
```
class Urun
    {
        public int Id;
        public string Ad;
        public double Fiyat;
        public static double KdvOran;
    }
```

KdvOran static tanımlandığından bir kez kullanıldı. Eğer 1000 ürün olsaydı kullanımın ne kadar önemli olduğu daha net anlaşılır.

```
class Program
    {
        static void Main(string[] args)
        {
            Urun urun1 = new Urun();
            urun1.Id = 1;
            urun1.Ad = "Monitör";
            urun1.Fiyat = 200;

            Urun urun2 = new Urun();
            urun2.Id = 2;
            urun2.Ad = "Klavye";
            urun2.Fiyat = 80;

            Urun.KdvOran = 0.18;
        }
    }
```

Static Olmak, Non-Static Olmak





const ve readonly Elemanlar

- Değişkenler konusunda gördüğümüz **const** anahtar sözcüğü özelliklerle de kullanılabilir.
- const olarak tanımlanan özellikler aynı zamanda static'tir. Dolayısıyla const özellikleri tekrar static anahtar sözcüğüyle belirtmek hatalıdır. const özellikler static özelliklerin taşıdığı tüm özellikleri taşır.
- const anahtar sözcüğü yalnızca object dışındaki özelliklerle (ve değişkenlerle) kullanılabilir. Dizilerle vs. kullanılamaz. const özellikler tanımlanır tanımlanmaz bir ilk değer verilmelidir.
- Sabit tanımlanmış üyeler de tıpkı statik tanımlılar gibi sınıfın genel elemanlarıdır ve nesne üzerinden erişilemez.
- Sabitlere erişmek için SınıfAdı.Sabit şeklinde kullanılırlar.

const ve readonly Elemanlar

```
using System;

class Matematik
{
    public const double PI = 3.14;
}

class Program
{
    static void Main()
    {
        Console.WriteLine(Matematik.PI);

        Matematik m = new Matematik();

        Console.WriteLine(m.PI);
    }
}
```

const ve readonly Elemanlar

- Referans tiplerin (object, diziler, nesneler) değerlerinin çalışma zamanında ayarlanmasından dolayı, referans tipler sabit olamaz.

- String veriler bu kuralın dışındadırlar. Yani const ile tanımlanabilirler.

- Referans tipleri sabit olarak tanımlamak için "readonly" anahtar sözcüğü kullanılır.

const ve readonly Elemanlar

```
using System;
class Sinif
{
   readonly int[] a={1,5,8};
   readonly object b=5;
   static void Main()
  {
   Sinif n=new Sinif();
   Console.WriteLine(n.a[0]);
   Console.WriteLine(n.b);
  }
}
```

const ve readonly Elemanlar

- Statik ifadelere de readonly eklenirse referans tipleri const gibi tanımlanmış olur.
- Static readonly ve const olarak tanımlanmış değişkenlerin tek farkı ilk değer verilme zamanında ortaya çıkar.
- const değişkenlerine derleme zamanında static readonly değişkenlerine ise çalışma zamanında ilk değer verilir.

const ve readonly Elemanlar

- Şimdi bir de salt okunur nesne oluşturalım:
- `using System;`
- `class Sinif`
- `{`
- `int a=5;`
- `static readonly Sinif nesne=new Sinif();`
- `static void Main()`
- `{ Console.WriteLine(nesne.a); }`
- `}`
- `//`Şimdilik nesnelerin salt okunur olması saçma gelebilir. Çünkü salt okunur nesnelerle ulaştığımız özellikleri değiştirebiliriz. Salt okunur yaptığımız nesnenin kendisidir.

const ve readonly Elemanlar

- readonly ile ilgili önemli bilgiler:

- readonly anahtar sözcüğü bir metot bloğunun içinde kullanılamaz.
- readonly anahtar sözcüğü bir nesne için kullanılacaksa static sözcüğü de kullanılmalıdır.
- readonly ve static sözcüklerinin sırası önemli değildir.
- readonly bir dizi ya da object türü ayrıca static olarak belirtilebilir.
- readonly anahtar sözcüğü const' un kullanılabildiği int gibi türlerle de kullanılabilir.
- readonly'nin constun aksine özellikleri staticleştirme özelliği yoktur.
- static sözcüğü de readonly gibi bir metot içinde kullanılamaz.

Visual Studio.Net -C#
5. HAFTA
Operatör Overloading, İndeksleyiciler, Yapılar, Enum sabitleri, İsim alanları

Örnek Uygulamalar

Örnek

```
using System;

class metotlar_3
{
    static int BuyukBul(int a, int b)
    {
        if (a > b)
            return a;
        return b;
    }

    static void Main()
    {
        int s1, s2;
        Console.Write("1. Sayıyı Girin:");
        s1 = Convert.ToInt32(Console.ReadLine());
        Console.Write("2. Sayıyı Girin:");
        s2 = Convert.ToInt32(Console.ReadLine());

        int enbuyuk = BuyukBul(s1, s2);

        Console.WriteLine("En büyük: {0}\'dir.", enbuyuk);
    }
}
```

Örnek

- Bir metodun geri dönüş değerini aynı metot için parametre olarak kullanabiliriz.

```
using System;

class metotlar_4
{
    static int BuyukBul(int a, int b)
    {
        if (a > b)
            return a;
        return b;
    }

    static int BuyukBul3(int a, int b, int c)
    {
        return BuyukBul(a, BuyukBul(b, c));
    }

    static void Main()
    {
        int s1, s2, s3;
        Console.Write("1. Sayıyı Girin:");
        s1 = Convert.ToInt32(Console.ReadLine());
        Console.Write("2. Sayıyı Girin:");
        s2 = Convert.ToInt32(Console.ReadLine());
        Console.Write("3. Sayıyı Girin:");
        s3 = Convert.ToInt32(Console.ReadLine());

        int enbuyuk = BuyukBul3(s1, s2, s3);

        Console.WriteLine("En büyük: {0}\'dir.", enbuyuk);
    }
}
```

Metot Parametresi Olarak Diziler

```
using System;
class metotlar_diziparametre
{
    static void DiziYaz(int[] a, int Sekil)
    {
        if (Sekil == 0)
        {
            foreach (Object o in a)
                Console.Write(o.ToString() + " ");
            Console.WriteLine();
        }
        else if (Sekil == 1)
        {
            int x=1;
            foreach (Object o in a)
            {
                Console.Write("{0,5}",o.ToString());
                if (x % 3 == 0) Console.WriteLine();
                x++;
            }
        }
        else
        {
            foreach (Object o in a)
                Console.WriteLine(o.ToString());
        }
        Console.WriteLine();
    }

    static void Main()
    {
        int[] dizi = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
        DiziYaz(dizi, 0);
        DiziYaz(dizi, 1);
        DiziYaz(dizi, 2);
    }
}
```

Örnek

```
using System;

class metotlar_5a
{
    static void DegerTipAktarim(int Sayi)
    {
        Sayi = 30;
    }

    static void Main()
    {
        int x = 100;
        Console.WriteLine(x);

        DegerTipAktarim(x);
        Console.WriteLine(x);
    }
}
```

- x değişkeninin değerini metot içinde değiştirmek değerini değiştirmemiştir. Çünkü x değer olarak metoda aktarılmıştır.

Metotların Aşırı Yüklenmesi Örnek

```
using System;
class Metotlar
{   static void Metot1(float x,float y)
  { Console.WriteLine("1. metot çağrıldı.");
  }
  static void Metot1(double x,double y)
  { Console.WriteLine("2. metot çağrıldı.");
  }
  static void Main()
  {   Metot1(5,6);   }
}
```

- Bu programda iki metodun da parametre sayısı eşit, iki metotta da tam tür uyumu yok ve iki metotta da bilinçsiz tür dönüşümü mümkün. Bu durumda en az kapasiteli türlü metot çağrılır. Yani bu programda birinci metot çağrılır.

```
using System;
class Metotlar
{   static void Metot1(float x,float y)
  { Console.WriteLine("1. metot çağrıldı.");
  }
  static void Metot1(int x,int y)
  { Console.WriteLine("2. metot çağrıldı.");
  }
  static void Main()
  {    Metot1(5,6.4f);   }
}
```

- Metot1 çağrılır.

Metotların Aşırı Yüklenmesi Örnek

```
using System;
class Metotlar
{
  static void Metot1(float x,float y)
  {  Console.WriteLine("1. metot çağrıldı.");
  }
  static void Metot1(int x,int y)
  {  Console.WriteLine("2. metot çağrıldı.");
  }
  static void Main()
  {       Metot1('f','g');   }
}
```

- Bu durumda ikinci metot çağrılır. Çünkü char hem int e hem de float a bilinçsiz olarak dönüşebilir. Ancak int daha az kapasitelidir.

```
using System;
class Metotlar
{ static void Metot1(int x,int y,int z)
  {  Console.WriteLine("1. metot çağrıldı.");  }
  static void Metot1(int x,int y)
  {  Console.WriteLine("2. metot çağrıldı.");  }
  static void Metot1(float x,int y)
  {   Console.WriteLine("3. metot çağrıldı."); }
  static void Main()
  {    Metot1(3,3,6);
       Metot1(3.4f,3);
       Metot1(1,'h');   }
}
```

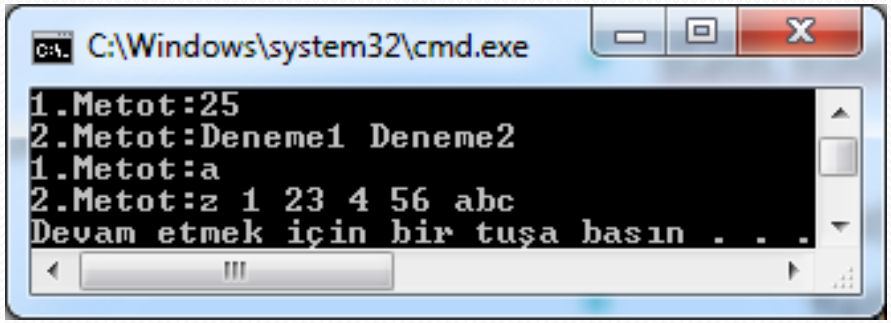
- Burada sırasıyla 1., 3. ve 2. metotlar çağrılacaktır.

Değişken Sayıda Parametre Alan Metotlar Örnek

```
using System;
class Metotlar
{   static int Islem(string a, params int[] sayilar)
   {   if (a == "carp")
      {   if (sayilar.Length == 0) return 1;
         int carpim = 1;
         foreach (int i in sayilar) carpim *= i;
         return carpim;
      }
      else if (a == "topla")
      {   if (sayilar.Length == 0) return 0;
         int toplam = 0;
         foreach (int i in sayilar) toplam += i;
         return toplam;
      }
      else return 0;
   }
   static void Main()
   {   Console.WriteLine(Islem("topla", 3, 4, 7, 8));
      Console.WriteLine(Islem("carp", 5, 23, 6));
   }
}
```

Değişken Sayıda Parametre Alan Metotlar Örnek

```
using System;
namespace params_ornek{
 class Program    {
   static void Yaz(object o)
   {
    Console.WriteLine("1.Metot:" + o);
   }
  static void Yaz(params object[] o)
  {
   if (o.Length == 0) return;
   Console.Write("2.Metot:");
   foreach (object n in o)
    Console.Write(n.ToString() + " ");
   Console.WriteLine();
  }
  static void Main(string[] args)
  {
   Yaz(25);
   Yaz("Deneme1", "Deneme2");
   Yaz('a');
   Yaz('z', 1, 23f, 4, 56, "abc");
  }
 }
}
```



Kendini çağıran (Recursive) metotlar Örnek

```
using System;
class faktoriyel {
        static int Faktoriyel(int a)       {
                if(a==0)
                return 1;
                return a*Faktoriyel(a-1); //fonksiyon kendi içerisinde çağrıldı
           }
        static void Main()    {
        Console.Write("Faktöriyelini alacağınız sayıyı giriniz :");
        int fak_değeri=Convert.ToInt32(Console.ReadLine());
        Console.WriteLine();
        Console.WriteLine("Girmiş olduğunuz sayının faktöriyeli :"+
                                Faktoriyel(fak_değeri));
        }
}
```

Örnek

- Programlamadaki metot kavramı aslında matematikteki fonksiyonlar konusunun aynısıdır. Matematikteki fonksiyonlar konusunda öğrendiğiniz bütün kuralları metotlarda uygulayabilirsiniz.
- Örneğin: Bir fonksiyona parametre olarak o fonksiyonun tersini verirsek sonuç x çıkacaktır. Örneğin f(x)=2x+5 olsun. f(x) fonksiyonunun tersi $f^{-1}(x)=(x-5)/2$'dir. Şimdi f(x) fonksiyonuna parametre olarak (x-5)/2 verirsek sonuç x olacaktır. Yani $f(x) \circ f^{-1}(x)=x$'tir. Şimdi bu kuralı bir metotla doğrulayalım.
- `using System;`
- `class Metotlar {`
- `static float Fonksiyon(float x) { return 2*x+5; }`
- `static float TersFonksiyon(float x) { return (x-5)/2; }`
- `static void Main() { float x=10;`
- `Console.WriteLine(Fonksiyon(x));`
- `Console.WriteLine(TersFonksiyon(x));`
- `Console.WriteLine(Fonksiyon(TersFonksiyon(x))); } }`

- Bu program ekrana sırasıyla 25, 2.5 ve 10 yazacaktır.

Sınıflar
Örnek

```
using System;

class Ogrenci
{
   public ulong OgrenciNo;
   public string Ad;
   public string Soyad;
   public string Bolum;
   public byte Sinif;
}
class Program
{
    static void Main()
    {
        Ogrenci ogr = new Ogrenci();

        Console.WriteLine(ogr.OgrenciNo);
        Console.WriteLine(ogr.Ad);
        Console.WriteLine(ogr.Soyad);
        Console.WriteLine(ogr.Bolum);
        Console.WriteLine(ogr.Sinif);
    }
}
```

Birden Fazla Sınıf Nesnesi Tanımlama Örnek

```
using System;
class KrediHesabi
{
    public ulong HesapNo;
}
class AnaSinif
{
   static void Main()
   {
      KrediHesabi hesap1=new KrediHesabi();
      KrediHesabi hesap2=new KrediHesabi();
      hesap1.HesapNo=3456;
      hesap2.HesapNo=1111;
     Console.WriteLine(hesap1.HesapNo);
     Console.WriteLine(hesap2.HesapNo);
   }
}
```

Sınıf Bildirimi Örnek

- **Örnek:**
- **using** System;
- **class** SinifIsmi
- {**public int** ozellik1=55; // başlangıç değerleri verildi.
- **public string** ozellik2="deneme";
- **public float** ozellik3=123.78f;
- **public int** metot1(**int** a,**int** b)
- { **return** a+b; }
- **public void** metot2(**string** a)
- { Console.WriteLine(a); }
- }
- **class** EsasSinif {
- **static void** Main() {
- SinifIsmi nesne=new SinifIsmi(); Console.WriteLine(nesne.ozellik1); Console.WriteLine(nesne.ozellik2); Console.WriteLine(nesne.ozellik3); Console.WriteLine(nesne.metot1(2,5)); nesne.metot2("deneme"); } }

Örnekler

Programdaki hatanın nedenini bulunuz

```
using System;
class Dortgen
{
   public int En = 30; public int Boy = 23;
   public int Alan()
   { int Alan = En * Boy; return Alan; }
   static void Main()
   { yaz d1 = new yaz(); d1.Yaz(); }
}
class yaz
{
   public void Yaz()
   {
      Console.WriteLine("En:{0,5}", Dortgen.En);
      Console.WriteLine("Boy:{0,5}", Dortgen.Boy);
      Console.WriteLine("Alan:{0,5}", Dortgen.Alan());
   }
}
```

Örnekler

Programdaki hatanın nedenini bulunuz

```
using System;
class Dortgen
{
   public int En = 30; public int Boy = 23;
   public int Alan()
   { int Alan = En * Boy; return Alan; }
   static void Main()
   { yaz d1 = new yaz(); d1.Yaz(); }
}
class yaz
{
   public void Yaz()
   {
      Console.WriteLine("En:{0,5}", Dortgen.En);
      Console.WriteLine("Boy:{0,5}", Dortgen.Boy);
      Console.WriteLine("Alan:{0,5}", Dortgen.Alan());
   }
}
```

- Çünkü, static olmayan metotların gövdesinde sadece aynı sınıftaki özellik ve metotlar static olsun olmasın direkt olarak kullanılabilir.
- public static int En=30;
- public static int Boy=23;
- public static int Alan() { … }
- yazılmalıydı

- Static metotların gövdesinde ise aynı sınıftaki özellik ve metotlar static olmadan direk olarak kullanılamaz. Değikenler ya static olacak yada sınıfa erişim için bir nesne oluşturulacak.

Örnekler

Çıktılarını bulunuz

```
using System;
class Dortgen
{
   int En;
   int Boy;
   static void Main()
   {
      int En=50;
      int Boy=100;
      Console.WriteLine(En+"\n"+Boy);
   }
}
```

```
using System;
 class Dortgen
 {
    static int En=8;
    static int Boy=3;
    static void Main()
    {
       int En=50;
       int Boy=100;
       Console.WriteLine(En+"\n"+Boy);
    }
 }
```

Örnekler

Çıktılarını bulunuz

```
using System;
class Dortgen
{
   static int En=8;
   static int Boy=3;
   static void Main()
   {
      int Boy=100;
      Console.WriteLine(En+"\n"+Boy);
   }
}
```

```
using System;
class Dortgen
{   public  int En=8;
     int Boy=3;
    static void Main()
   {
   Console.WriteLine(En+"\n"+Boy);
   }
}
// hata verir. Çözüm yollarını söyleyiniz.
```

Static Üye konusunda cevaplarınızı gözden geçiriniz.

- 1.yol :Main başındaki static ifadesi kaldırılır(Program yine hata verir. Main mutlaka static olacak
- 2.yol:Değişkenler static yapılır
- 3.yol: sınıf için nesne oluşturularak değişkenlere erişilir.

Örnek

```
// 1 'den 100 e kadar sayıların toplamı
using System;
      class Topla
      {        public int tpl;

         public Topla(int toplam)
         {   tpl = toplam;     }

         public int Toplama
         {  get { return tpl; } // değeri görüntüleme
            set { tpl= value+tpl; } // yeni değeri depola
          }

      static void Main(string[] args)
   {    Topla ekle = new Topla(0);
        Console.WriteLine("Başlangıç değeri->" + ekle.Toplama);
         for (int i = 1; i < 100; i++)            ekle.Toplama = i;
         Console.WriteLine("Sayıların toplamı" + ekle.Toplama);
          }
     }
```

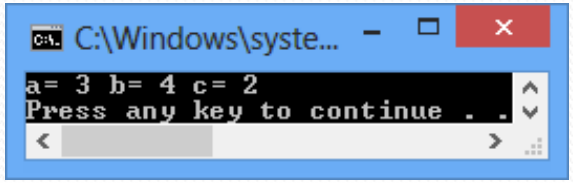
Yapıcı Metot (Default Constructor)-Örnek

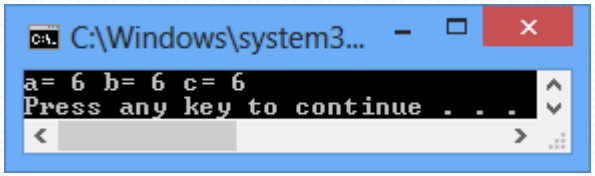
```
using System;

class Toplama
{
    public int X;
    public int Y;

    public Toplama(int x, int y)
    {
        X = x;
        Y = y;
    }

    public Toplama():this(-1,-1)
    {
    }

    public Toplama(int x):this(x, 1)
    {
    }

    public int Islem()
    {
        return X + Y;
    }

    public void Yaz()
    {
        Console.WriteLine("X = {0}", X);
        Console.WriteLine("Y = {0}", Y);
    }
}
```

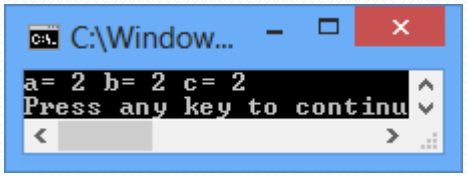
```
class Program
{
    static void Main()
    {
        Toplama t = new Toplama(5, 6);
        t.Yaz();
        Toplama y = new Toplama();
        y.Yaz();
        Toplama u = new Toplama(7);
        u.Yaz();
    }
}
```

Örnek: Random

```
class Program
{  // Random rnd = new Random(); //kullanılsaydı yine farklı değer üretirdi.
    public  int Zar()
    { Random rnd = new Random();      return rnd.Next(1,7); }
    public static void metotlar(int a, int b, int c)
    { Console.WriteLine("a= " + a + " b= " + b + " c= " + c); }
    static void Main(string[] args)
    {  Program a = new Program();
       metotlar(a.Zar(), a.Zar(), a.Zar());
    }
}
```







- Random, arka arkaya varsayılan yapıcı metot çağrısı kullanılarak oluşturulan nesneler aynı varsayılan değerler olacaktır (sistem saati aynı değer olduğundan) ve bu nedenle, aynı rasgele sayılar kümesi oluşturacaktır. Kümeden seçilen değerler aynı değer olacaktır.
- /*static void Main(string[] args)
- { Random rnd = new Random(); Program a = new Program();
- metotlar(rnd.Next(1,7), rnd.Next(1,7), rnd.Next(1,7)); } şeklinde kullansaydık her defasında farklı değer üretirdi.