Sınıflar
(Classes)

- Çözülmesi istenene problemi çeşitli parçalara ayırıp her bir parça arasındaki ilişkiyi gerçeğine uygun bir şekilde belirleme tekniğine nesne yönelimi denir.

- Bu parçalar arasındaki ilişkiyi kullanıp büyük çaplı programlar geliştirme tekniğine de nesne yönelimli programlama denir.

- Nesnelerin kaynak kodundaki karşılığı sınıflardır.

Sınıflar(Classes)

- C# dilinin ve nesne yönelimli programlama tekniğinin en önemli veri yapısıdır. Sınıflar bizlere bir veri modeli sunar.
- Sınıfların üye elemanları değişkenler(özellik) ve metotlardır.
- Sınıf bildirimleri class anahtar sözcüğü kullanılarak yapılır.

```
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
        }
}
```

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

C#’da 5 tane erişim belirleyici bulunmaktadır:

a. **public**
b. **private**
c. protected
d. internal
e. internal protected

- “public” erişim belirteci ile tanımlanan sınıfın bir özelliğine ya da metoduna istenilen yerden erişime izin verilmiş olur. public üye elemanlar genelde sınıfın dışa sunduğu arayüzü 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.

- “internal” kodun bulunduğu yere bağlı olarak “public” ya da “protected” gibi davranabilen bir erişim belirtecidir.

Sınıf Bildirimi

- Sınıflar birer veri yapısıdır, gerçek hayattaki herhangi bir nesne bu yapı ile modellenebilir.

- Sınıf tanımlamaları yapılırken amaca uygun isimler seçilmelidir.

```
class Ogrenci
{
        public ulong OgrenciNo;
        public string Ad;
        public string Soyad;
        public string Bolum;
        public byte Sinif;
}
```

- 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.

- Sınıf bildirimleri için bellekte yer tahsis edilmez. Sınıf bildirimleri sonradan tanımlanacak olan nesneler için derleyiciye nesnenin neye benzeyeceğini gösterir.

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. Bunun için de;

  Ogrenci ogr1 = new Ogrenci();

  kullanılır.

```
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);
    }
}
```

- 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.

```
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();

        ogr.OgrenciNo = 123456789;
        ogr.Ad = "Ahmet";
        ogr.Soyad = "Yıldız";
        ogr.Bolum = "Bilgisayar Mühendisliği";
        ogr.Sinif = 3;

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

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

Sınıf tanımlaması şu şekilde olsaydı?

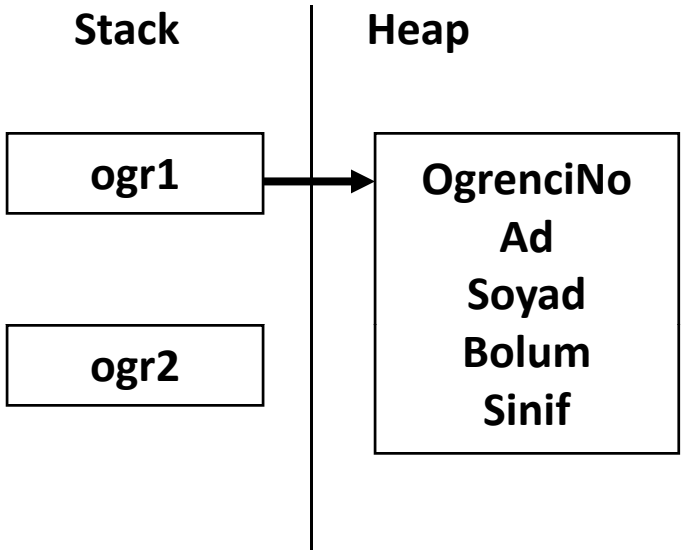
```
class Ogrenci
{
  public ulong OgrenciNo;
  string Ad;
  string Soyad;
  private string Bolum;
  public byte Sinif;
}
```

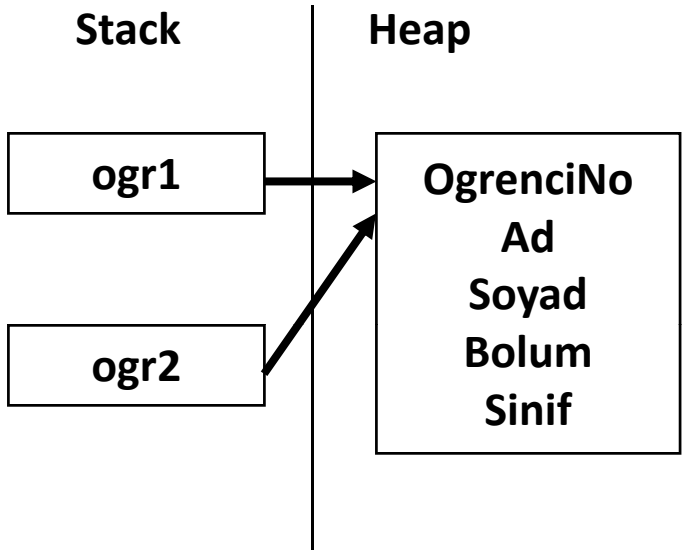
Birden Fazla Sınıf Nesnesi Tanımlama

- Bir sınıftan birden fazla nesne de tanımlanabilir. Fakat bu tanımlanmış nesneler ile işlem yapılırken referans tiplerde anlatılan kurallara dikkat edilmelidir.

- 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.

```
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 ogr1 = new Ogrenci();
        Ogrenci ogr2;
        ogr1.OgrenciNo = 123456789;
        ogr1.Ad = "Ahmet";
        ogr1.Soyad = "Yıldız";
        ogr1.Bolum = "Bilgisayar Mühendisliği";
        ogr1.Sinif = 3;
        ogr2 = ogr1;
        Console.WriteLine(ogr2.OgrenciNo);
        ogr1.Ad="Hamdi";
        Console.WriteLine(ogr1.Ad);
        Console.WriteLine(ogr2.Soyad);
        Console.WriteLine(ogr2.Bolum);
        Console.WriteLine(ogr2.Sinif);
    }
}
```






```
Ogrenci ogr1 = new Ogrenci();
Ogrenci ogr2;
```

```
ogr2 = ogr1;
```

Sınıflara Metot Ekleme

- Sınıfın üyeleri yani sınıfı oluşturan temel elemanlar özellikler (değişkenler) ve metotlardır.
- Metotlar sınıf içersinde çeşitli işlemleri gerçekleştiren yapılardır. Metotlar konusunda işlendiği gibi tanımlanırlar.
- Erişim belirteçleri sayesinde sınıf içersinden ya da dışarısından da kullanılabilirler.

Dikdörtgen Sınıfı

```
using System;

class Dikdortgen
{
    public int UzunKenar;
    public int KisaKenar;

    public void DegerBelirle(int uzunkenar, int kisakenar)
    {
        UzunKenar = uzunkenar;
        KisaKenar = kisakenar;
    }

    public int Cevre()
    {
        return 2 * UzunKenar + 2 * KisaKenar;
    }

    public int Alan()
    {
        return UzunKenar * KisaKenar;
    }
}
```

Dikdörtgen Sınıfının Yaz() Metodu

```
public void Yaz()
    {
        Console.WriteLine("************************");
        Console.WriteLine("* Uzun Kenar = {0,5} *",UzunKenar);
        Console.WriteLine("* Kısa Kenar = {0,5} *",KisaKenar);
        Console.WriteLine("* Çevre      = {0,5} *",Cevre());
        Console.WriteLine("* Alan       = {0,5} *",Alan());
        Console.WriteLine("************************\n");
    }
```

Dikdörtgen Sınıfını Kullanan Başka Bir Sınıf

```
class Program
{
    static void Main()
    {
        Dikdortgen d1 = new Dikdortgen();

        d1.DegerBelirle(40, 25);
        d1.Yaz();

        Dikdortgen d2 = new Dikdortgen();

        d2.DegerBelirle(15, 12);
        d2.Yaz();

        Dikdortgen d3 = new Dikdortgen();
        d3.UzunKenar = 78;
        d3.KisaKenar = 55;
        d3.Yaz();
    }
}
```

Negatif Değerlerde Düzeltme Yapan Yeni Bir Metodun Eklenmesi

```
public void DegerBelirle(int uzunkenar, int kisakenar)
    {
        if (uzunkenar <= 0 || kisakenar <= 0)
        {
            UzunKenar = 0;
            KisaKenar = 0;
        }
        else
        {
            UzunKenar = uzunkenar;
            KisaKenar = kisakenar;
        }
    }
```

this Anahtar Sözcüğü

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

- Bir çok 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.

```
using System;

class Toplama
{
    public int x;
    public int y;

    public void DegerBelirle(int x, int y)
    {
        x = x;
        y = y;
    }

    public void Topla()
    {
        Console.WriteLine(x + y);
    }
}

class Program
{
    static void Main()
    {
        Toplama t = new Toplama();
        t.DegerBelirle(15,20);
        t.Topla();
    }
}
```

set ve get Anahtar Sözcükleri

- Bir nesnenin üye değişkenlerine değer atarken ya da içersindeki 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.

```
using System;

class Toplama
{
    private int mX;
    private int mY;

    public int x
    {
        get
        {
            Console.WriteLine("x get
                  çalıştı...");
            return mX;
        }
        set
        {
            Console.WriteLine("x set
                  çalıştı...");
            mX = value;
        }
    }
    public int y
    {
        get
        {
            Console.WriteLine("y get
                  çalıştı...");
            return mY;
        }
        set
        {
            Console.WriteLine("y set
                  çalıştı...");
            mY = value;
        }
    }

    public int Topla()
    {
        return mX + mY;
    }
}

class Program
{
    static void Main()
    {
        Toplama t = new Toplama();

        t.x = 25;
        t.y = 15;
        Console.WriteLine(t.Topla());

        int a = t.y;
        t.y = t.x;
        t.x = a;
    }
}
```

- Görüldüğü gibi, değişkenin değerini değiştirdiğimizde çalışmasını istediğimiz kodları set blokları arasına yazarız.

- Verilen örnekte yeni bir anahtar sözcük olan value vardır. Bu anahtar sözcük, özelliğe atanacak nesnenin değerini ifade eder.

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.

Yapıcı Metotlar(Constructors)

- Bir nesne 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.

- Yapıcı metot tanımlanırken dikkat edilmesi gerek iki önemli nokta vardır. Bunlardan birincisi yapıcı metotlarda geri dönüş değeri diye bir kavram yoktur. (Fakat geri dönüş değerinin olmaması void olarak tanımlanması anlamına gelmez.)

- Diğeri ise yapıcı metotların isimleri sınıf ismi ile aynı olmalıdır.

- Yapıcı metotlar dışarıdan çağrıldığı için public olarak tanımlanırlar.

```
using System;

class Toplama
{
    public int X;
    public int Y;

    public Toplama(int x,int y)
    {
        Console.WriteLine("Yapıcı metot çalıştı...");
        X = x;
        Y = y;
    }

    public int Islem()
    {
        return X + Y;
    }
}
class Program
{
    static void Main()
    {
        Toplama t = new Toplama(25,50);
        //Console.WriteLine(t.Islem());
    }
}
```

Varsayılan Yapıcı Metot (Default Constructor)

- Otomatik olarak tanımlanmış sınıf nesneleri oluşturulduğu anda çağrılan yapıcı metottur. Her sınıf nesnesinin en az bir yapıcı metodu bulunur.

- Varsayılan yapıcı metodun parametresi yoktur. Nesnenin üye elemanlarına varsayılan değerlerini atar.

- Eğer kendi yapıcı metodumuzu tanımlarsak varsayılan yapıcı metotlar çalıştırılmaz.

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

- Aşırı yükleme this anahtar sözcüğü kullanılarak da oluşturulabilir.

```
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();
    }
}
```

Kopyalayıcı Yapıcı Metot

- 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();
    }
}
```

Yıkıcılar (Destructors) ve Dispose() Metodu

- Nesnelere erişimin mümkün olmadığı zamanlarda nesnelerin bellekte kalmasının bir anlamı yoktur.

- C# dilinde gereksiz nesneleri kontrol eden ve bunların kullandıkları belleği zamanı gelince iade eden (Garbage Collection) bir mekanizma bulunur.

- Gereksiz nesne toplayıcısı (Garbage Collector) programcı yerine gereksiz nesnelerin kullandıkları bellek alanını iade eder.

- Yalnız bu sistem nesnelerin yok edilmesi ve belleğin iadesinin, faaliyet alanının sona erdiği noktada yapılacağını garanti etmez. Yani nesnenin ne zaman yok edileceği kesin bilinemez.

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

Dispose( ) Metodu

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

- IDisposable arayüzü kullanılarak yapılır.

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

```
using System;

class Yikici
{
    ~Yikici()
    {
        Console.WriteLine("Yıkıcı metot çalıştı...");
    }

}
class Program
{
    static void Main()
    {
        {
            Yikici y = new Yikici();
        }
        Console.WriteLine("Ana Program...");
    }
}
```

- 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.

```
using System;

class Yikici
{
    ~Yikici()
    {
        Console.WriteLine("Yıkıcı metot çalıştı...");
    }

}
class Program
{
    static void Main()
    {
        {
            Yikici y = new Yikici();
        }
        GC.Collect();
        Console.Read();
        Console.WriteLine("Ana Program...");
    }
}
```

- 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.

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.

```
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));
    }
}
```

- Main() metodu da herhangi bir nesneye ihtiyaç duymadan çalışabilmesi için static olarak tanımlanmaktadır.

- 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.

```
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();
    }
}
```

```
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.

```
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();
            Console.WriteLine(b.deneme());
            Console.WriteLine(Statik.a);
            Console.ReadLine();
        }
    }
}
```

Statik Yapıcı Metotlar

- 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 değiştirmem mümkün değildir.
- Statik yapıcı metotlar ilk nesne tanımlandığında çalıştırılır.

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. Aynı şekilde static yapıcı metod da tanımlanamaz.

- Daha sonra bahsedilecek olan nesne konusundaki türeme başlığında tekrar bahsedeceğimiz üzere; static sınıflar türetmeyi desteklemez.

const ve readonly Elemanlar

- Değerinin değişmesi istenmeyen bir üye değişken sabit olarak tanımlanabilir.

- Sabitler const anahtar sözcüğü ile tanımlanır. Mutlaka ilk değer verilmesi gerekir ya da derleme esnasında mutlaka hesaplanabilmelidir.

```
public const double PI = 3.14;

int b = 10; const a = b * 2;
```

- Sabit tanımlanmış üyeler de tıpkı statik tanımlılar gibi sınıfın genel elemanlarıdır ve nesne üzerinden erişilemez.

- Sabitlere erişmek için SınıfAdı.Sabit şeklinde kullanılırlar.

```
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);
    }
}
```

- Referans tiplerin 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.

- Statik ifadelere readonly eklenirse referans tipleri const gibi tanımlanmış olur.

- Static ve 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.

Operatör Aşırı Yükeleme (Operator OverLoading)

- Nesneye yönelik programlama mantığında yazılan sınıflar için operatörlere yeni anlamlar yüklenebilir.

- Bu işlem operatör metotları ile gerçekleştirilir.

  Operatör metotları bir nesnenin ifadeler içinde operatörle kullanıldığında davranışını belirler.Operatör metotları klasik metotlar gibidir.

Operatör Metotları Tanımlanırken Uyulması Gereken Kurallar

- Operatör metotları static olarak tanımlanmalıdır.
- Operatör metotları isimlerinde "operator" anahtar sözcüğü kullanılmalıdır. (operator+, operator* gibi)
- Bütün operatör metotları tek ya da iki parametre almalıdır.

- Operatör metotları da aşırı yüklenebilir.
- Tekli (unary) operatör metotolarında parametre mutlaka sınıf türünden olmalıdır. İkili (binary) operatörlerde ise en az bir parametre ilgili sınıf türünden olmalıdır.
- Operatör metotları ref ve out anahtar sözcükleri kullanmamalıdır.

```
using System;

class KarmasikSayi
{
    private double mGercek;
    private double mSanal;

    public double Gercek
    {
        get { return mGercek; }
        set { mGercek = value; }
    }

    public double Sanal
    {
        get { return mSanal; }
        set { mSanal = value; }
    }

    public KarmasikSayi(double x, double y)
    {
        mGercek = x;
        mSanal = y;
    }

    public KarmasikSayi()
    {
        mGercek = 0;
        mSanal = 0;
    }
```

```
 public KarmasikSayi(KarmasikSayi k)
    {
        mGercek = k.mGercek;
        mSanal = k.mSanal;
    }

    public void Yaz()
    {
        if (mSanal > 0)
            Console.WriteLine("{0}+{1}i",
                                          mGercek,
    mSanal);
        else
            Console.WriteLine("{0}-{1}i",
                                          mGercek,
    -mSanal);
    }
}

class Program
{
    public static void Main()
    {
        KarmasikSayi k = new KarmasikSayi(-5,-6);
        k.Yaz();
    }
}
```

```
public static KarmasikSayi operator +(KarmasikSayi a,
  KarmasikSayi b)
    {
        double gt = a.Gercek + b.Gercek;
        double st = a.Sanal + b.Sanal;
        return new KarmasikSayi(gt, st);
    }
```

```
class Program
{
    public static void Main()
    {
        KarmasikSayi k1 = new KarmasikSayi(-5,-6);
        KarmasikSayi k2 = new KarmasikSayi(4, 7);
        KarmasikSayi t = k1 + k2;
        t.Yaz();
    }
}
```

```
 public static KarmasikSayi operator +(KarmasikSayi a, double b)
    {
        double gt = a.Gercek + b;
        return new KarmasikSayi(gt, a.Sanal);
    }

    public static KarmasikSayi operator +(double b, KarmasikSayi a)
    {
        return a + b;
    }
```

```
class Program
{
    public static void Main()
    {
        KarmasikSayi k1 = new KarmasikSayi(5,-5);
        KarmasikSayi t = 10 + k1;
        t.Yaz();
        KarmasikSayi y = k1 + 7;
        y.Yaz();
        KarmasikSayi z = k1 + k1;
        z.Yaz();
    }
}
```

Karmaşık Sayı Sınıfı için;

- Çıkarma
- Çarpma
- Bölme
- Karşılaştırma (eşitlik)

Operatörlerini aşırı yükleme metotlarını yazınız.

- Aritmetiksel operatörlerin dışında ilişkisel operatörler, mantıksal operatörler ve dönüşüm operatörleri de aşırı yüklenebilir.

- C# dilinde işlemli atama operatörleri , &&, ||, [], (), =, ? , ?: , -> , is , sizeof, new, typeof gibi operatörler aşırı yüklenemez.

- Temel veri türleri için operatörler yeniden yüklenemez.

- Operatörlerin var olan öncelik sırası da değiştirilemez.