Operator Aşırı Yükleme
(Operator OverLoading)

Operator Aşırı Yükleme

- Operatör metotları bir nesnenin ifadeler içinde operatörlerle kullanıldığı zaman davranışını belirler.
- Temel veri türleri için operatörler zaten belirlenmiştir. Dolayısıyla bunlara söz konusu olması mümkün değildir. Fakat kendi tasarladığımız sınıflar için operatörlere yeni anlamlar yüklememiz mümkündür.

- Operatör metotları tanımlanırken dikkat edilmesi gereken bazı 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.

Örnek

- Karmaşık sayılara ait olmak üzere tanımlanan KarmaşıkSayi sınıfından çeşitli karmaşık sayı nesneleri üretilecektir ve bu nesneler üzerinde aritmetiksel ve mantıksal işlemler yapan çeşitli operatör metotları tanımlanacak ve bunlar aşırı yüklenecektir.

```
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();
    }
}
```

Aritmetiksel Operatörlerin Aşırı Yüklenmesi

- Tanımlanacak olan operator+ metodu karmaşık sayılar üzerinde toplamı işlemi yaparak ve sonucu yeni bir kompleks sayı nesnesi olarak döndürmektedir.

```
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();
    }
}
```

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

- Yukarıda tanımlanmış operatör metodu ile aşağıdaki ifadeler de geçerli kılınmış olur :

  KarmasikSayi a =new KarmasikSayi(-3,6);

  KarmasikSayi c = a+5;

  KarmasikSayi d = a+5d;

  KarmasikSayi e = a+ 5f;

  ~~KarmasikSayi e = 5+a;~~

- Toplama operator metoduna benzer şekilde çıkarma operatör metodu da kolaylıkla tanımlanabilir.

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

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

```
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();
    }
}
```

- Çarpma ve bölme işlemleri için de aynı yaklaşımla operatör metodları oluşturalım.

```
public static KarmasikSayi operator *(KarmasikSayi a,
  KarmasikSayi b)
    {
        double sanal1 = a.Gercek * b.Sanal;
        double sanal2 = a.Sanal * b.Gercek;
        double sc = sanal1 + sanal2;

        double gercek1 = a.Gercek * b.Gercek;
        double gercek2 = a.Sanal * b.Sanal;
        double gc = gercek1 - gercek2;

        return new KarmasikSayi(gc, sc);
    }
```

```
public static KarmasikSayi operator /(KarmasikSayi a,
 KarmasikSayi b)
   {
 KarmasikSayi beslenik = new KarmasikSayi(b.Gercek, -
 b.Sanal);

 KarmasikSayi pay = a * beslenik
 double payda = b.Gercek * b.Gercek + b. Sanal * b.
 Sanal);

 double bolumGercek = pay.Gercek / payda;
 double bolumSanal = pay.Sanal / payda;

 return new KarmasikSayi(bolumGercek, bolumSanal);
   }
```

İlişkisel Operatörlerin Aşırı Yüklenmesi

- İlişkisel operatör metotları, true veya false değer ile geri dönerler. İlişkisel operatörler daha önceden de hatırlanacağı gibi 6 tanedir (!= , == , < , > , <= , >=)

- Bu operatörlerin aşırı yüklenmesinde dikkat edilecek tek husus zıt anlamlı operatörlerin her ikisinin de aynı anda yüklenmiş olması gerektiğidir.

```
public static bool operator ==(KarmasikSayi a,
  KarmasikSayi b)
    {
  if (a.Sanal == b.Sanal && a.Gercek == b.Gercek)
  return true;
  else return false;
    }


public static bool operator !=(KarmasikSayi a,
  KarmasikSayi b)
    {
  return !(a==b);
    }
```

true ve false Operatörlerin Aşırı Yüklenmesi

- Bu iki operatör de aynı anda olmak koşuluyla aşırı yüklenebilirler. Eğer bu operatörler aşırı yüklenirlerse koşul ifadelerini aşağıdaki gibi kullanmak mümkün olur :

```
if (kompleks)
 {……..}
else
{……..}
```

```
public static bool operator true(KarmasikSayi a)
    {
  if (a.Sanal != 0 || a.Gercek != 0)
  return true;
  else return false;
    }

public static bool operator false(KarmasikSayi a)
    {
  if (a.Sanal == 0 || a.Gercek == 0)
  return true;
  else return false;


    }
```

Mantıksal Operatörlerin Aşırı Yüklenmesi

- Mantıksal operatör metotlarının aşırı yüklenmesini 2 kısımda incelemek mümkündür:
  - &, |, ! ve (birinci grup)
  - && ve || operatörleri (ikinci grup)

- && ve || operatörleri direkt olarak aşırı yüklenemezler. Baz ön şartları vardır.

```
public static bool operator |(KarmasikSayi
  a,KarmasikSayi b)
    {
  if ((a.Sanal != 0 || a.Gercek != 0) | ( b.Sanal!= 0
  || b.Gercek !=0))
  return true;
  else return false;
    }


public static bool operator &(KarmasikSayi
  a,KarmasikSayi b)
    {
  if ((a.Sanal == 0 || a.Gercek == 0) | ( b.Sanal== 0
  || b.Gercek ==0))
  return false;
  else return true;
    }
```

```
public static bool operator !(KarmasikSayi a)
    {
  if ((a.Gercek != 0) | ( a.Sanal!= 0 ))
  return false;
  else return true;
    }
```

Mantıksal operatör metotlarının aşırı yüklenmesi için birtakım şartların oluşması gerekir:

- & ve | operatörleri aşırı yüklenmiş olmalıdır.
- true ve false operatörlerinin aşırı yüklenmiş olması gerekir.
- operator& ve operator| metotlarının parametreleri ilgili sınıfın türünden olmalıdır.
- operator& ve operator| metotlarının geri dönüş değeri ilgili sınıfın türünden olmalıdır.

Dönüşüm Operatörlerin Aşırı Yüklenmesi

- Tür dönüşüm operatörleri ile aşağıdaki bir ifadenin nasıl bir sonuç üretebileceğini bilebilriz.

```
KarmasikSayi k = new KarmasikSayi();
int a = k;
```

- Dönüşüm operatörleri aşırı yüklenmez ise, derleyici yukarıdaki gibi bir durumla karşılaştığında ne yapacağını bilemez.

```
public static implicit operator HedefTur(DonusturulecekTur)
    {
  return HedefTur
     }
```

veya

```
public static explicit operator HedefTur(DonusturulecekTur)
    {
  return HedefTur
     }
```

Örn:

Eğer k KarmasikSayi türünden bir nesne ise;

```
int a = k ; /* atamasının geçerli olabilrmesi için */
```

```
public static implicit operator int(KarmasikSayi k)
    {
  return 10; // return k.Gercek ifadesi de olabilirdi.
    }
```

Örn:

Eğer k KarmasikSayi türünden bir nesne ise;

```
int a = (int) k ; /* atamasının geçerli olabilrmesi için */
```

```
public static explicit operator int(KarmasikSayi k)
    {
  return 10; // return k.Gercek ifadesi de olabilirdi.
    }
```

Indeksleyiciler (Indexers), Yapı (Structs) ve Enum Türleri

- İndeksleme, dizi elemanlara erişmek için kullanılan “[]” operatöründen gelmektedir.

- System.Array sınıfından tanımlanan diziler aslında birer nesneydi ve tanımlanan indeksleyici sayesinde nesnenin elemanlarına “[]” operatörleri ile ulaşılabiliyor.

- İndeksleyiciler, olsaydı operator[] metodu yerine (böyle bir metot yoktur) faaliyet gösterirlerdi.

- İndeksleyiciler genelde sınıfın üye elemanlarından birisi dizi ya da benzer bir türden ise kullanışlı olur. Ancak bu amacın dışında da kullanılması tamamıyla serbesttir.

- Ancak operatörlerin taşıdıkları anlamlar çerçevesinde kullanılmaları daha doğru olur.İndeksleyiciler de diziler gibi tek yada çok boyutlu olabilir.

Tek Boyutlu İndeksleyici:

Genel Tanımlaması:

```
ElemanTipi this[IndeksTipi indeks]
{
   get { return değer; }
   set { işlemler; …;   }
}
```

```
using System;

class Indeksleyici
{
    public double sayi;

    public double this[double indeks]
    {
        get { return indeks * indeks; }
        set { sayi = value; }
    }
}

class Program
{
    public static void Main()
    {
        Indeksleyici i = new Indeksleyici();
        Console.WriteLine("i[1.6]={0}",i[1.6]);

        i[3] = 3;
        Console.WriteLine(i.sayi);

        i[7] = 7;
        Console.WriteLine(i.sayi);

        Console.WriteLine("i[6]={0}", i[6]);
    }
}
```

- İndeksleyicilerde indeks değeri dizilerde olduğu gibi tamsayı olmak zorunda değildir.

- Fakat ideal kullanımda tamsayı olarak seçilmelidirler.

- Örnekte indeks değeri double türündendir ve geri dönen değer sayi*sayi'dir. Yani indeksleyicilerin kullanım amacına uygun bir örnek değildir.

```
using System;

class Indeksleyici
{
    private int[] dizi;

    public Indeksleyici(int DiziUzunlugu)
    {
        dizi = new int[DiziUzunlugu];
    }

    public int DiziBoyutu
    {
        get { return dizi.Length; }
    }

    public int this[int indeks]
    {
        get { return dizi[indeks]; }
        set { dizi[indeks] = value; }
    }
}

class Program
{
    public static void Main()
    {
        Indeksleyici i = new Indeksleyici(10);

        for (int k = 0; k < i.DiziBoyutu; k++)
        {
            Console.WriteLine("i[{0}]={1}", k, i[k]);
        }
    }
}
```

- İndeksleyiciler daha çok sınıfın üye dizilerinden birine direkt erişmek için kullanılırlar.

- Diziler dinamik olarak oluşturulabilir.

- Nesne ismine indeks verilerek üye dizinin elemanının değeri elde edilir.

- İndeksleyiciler de aşırı yüklenebilir. Fakat bu çok kullanılan bir yöntem değildir. İndeksleyiciler çok boyutlu da tanımlanabilir.

Yapılar (Structs)

- C# dilinde yapılar değer tipindedir.
- Sınıf bildirimine çok benzer şekilde tanımlanırlar. struct anahtar sözcüğü kullanılır.

- Bazı durumlarda belli bir grup verinin bir arada tanımlanması için sınıfların kullanılması verimsiz olur. Sınıf kullanıldığında stack alanında referans tipte değişken oluşturulur ve üyeler içinde heap alanında ayrıca bellek ayrılır. Verilere ulaşmak için referans kullanmak istenmeyebilir.

- Bu gibi durumlarda az sayıda ve veri tipindeki değişkenleri yapı şeklinde tanımlamak hız ve verimlilik sağlayabilir.

Yapı Tanımlaması:

```
struct YapıAdı
{
   Elemanlar;
}
```

Şeklinde tanımlanır. Yapılar daha çok birbiri ile ilişkili değerleri bir araya toplamak için kullanılır.

- Yapılar değer tipidir ve yapı türündeki nesneler stack alanında saklanır.
- Yapılar da diğer tüm nesneler gibi object sınıfından türetilmiştir.
- Yapılar kalıtımı desteklemez, türetme yapılamaz.

```
struct Ogrenci
{
    public int Numara;
    public string Ad;
    public string Soyad;
}
```

- Yapılar tanımlandıktan sonra bir yapı nesnesi oluşturmak için yine new operatörü kullanılır.

- Varsayılan yapıcı metot ya da bizim belirlediğimiz yapıcı metot çalışarak ilk değer ataması yapar.

- Sınıflardan farklı olarak new kullanılmadan da yapılar tanımlanabilir.

- Bu şekilde tanımlanan yapı nesnelerinin üye elemanlarına ilk değer elle verilmelidir.

- Yapı nesnesinin elemanlarına sınıflarda olduğu gibi “.” ile erişilir.

```
using System;

class Yapilar
{
    struct Ogrenci
    {
        public int Numara;
        public string Ad;
        public string Soyad;
    }

    public static void Main()
    {
        Ogrenci ogr1;

        Console.WriteLine(ogr1.Numara);
    }
}
```

```
using System;

class Yapilar
{
    struct Ogrenci
    {
        public int Numara;
        public string Ad;
        public string Soyad;
    }

    public static void Main()
    {
        Ogrenci ogr1=new Ogrenci();

        ogr1.Numara = 123;
        ogr1.Ad = "Ali";
        ogr1.Soyad = "Türk";

        Ogrenci ogr2 = ogr1;

        ogr2.Numara = 456;

        Console.WriteLine("{0} {1} {2}", ogr1.Numara, ogr1.Ad, ogr1.Soyad);
        Console.WriteLine("{0} {1} {2}", ogr2.Numara, ogr2.Ad, ogr2.Soyad);
    }
}
```

```
using System;

class Yapilar
{
    public struct Ogrenci
    {
        public int Numara;
        public string Ad;
        public string Soyad;
    }

    public static void Metot(Ogrenci o)
    {
        o.Numara = 999;
    }

    public static void Main()
    {
        Ogrenci ogr=new Ogrenci();

        ogr.Numara = 123;
        ogr.Ad = "Ali";
        ogr.Soyad = "Türk";

        Metot(ogr);

        Console.WriteLine("{0} {1} {2}", ogr.Numara, ogr.Ad, ogr.Soyad);
    }
}
```

- Yapıların birden fazla yapıcı metotları olabilir. Ancak varsayılan yapıcı metodu biz bildiremeyiz.

- Bütün yapıcıların parametre alma zorunluluğu vardır.

- Yapıcı metot bildirildiği zaman bütün elemanlara ilk değer verme zorunluluğu vardır. Bu değer ataması mutlaka parametre ile olmak zorunda değildir.

```
using System;

class Yapilar
{
    public struct Ogrenci
    {
        public int Numara;
        public string Ad;
        public string Soyad;

        public Ogrenci(int no, string ad, string soyad)
        {
            Numara = no;
            Ad = ad;
            Soyad = soyad;
        }
    }

    public static void Main()
    {
        Ogrenci ogr = new Ogrenci(123,"Ali","Türk");

        Console.WriteLine("{0} {1} {2}", ogr.Numara, ogr.Ad,
                                                    ogr.Soyad);
    }
}
```

- Yapı nesneleri faaliyet alanı bitince otomatik olarak stack bölgesinden silinirler. Programcının nesnenin kaynaklarını geri almak için yapması gereken bir şey yoktur. Bu yüzden yapılarda yıkıcı metotlar bildirilemez.

- Sınıflarda olduğu gibi yapılar içinde değişkenler için set-get metotları ile indeksleyiciler kullanılabilir.

Avantajlar

- Yapıların bazı avantajları göz önünde bulundurularak belli problemlerin çözümünde kullanılabilirler:
  - Stack bellek alanında yer ayırma, kopyalama gibi işlemler heap alanından daha hızlıdır.
  - Sınıf nesneleri Garbage Collection mekanizması ile heap alanından silindikleri için yıkıcı metotların ne zaman çağrılacağı kesin bilinemez. Yapılarda yıkıcı metot olmamasına rağmen faaliyet alanı bitiminde bellekten otomatik silinirler.

Numaralandırma (Enumeration)

- Numaralandırma (enumeration) belli bir sayısal değer alan toplu sabitler tanımlamak için kullanılan bir yapıdır.

  enum anahtar sözcüğü ile tanımlanır.

- enum sabitleri çeşitli sembolleri sayılarla ifade etmek için geliştirilmiş bir veri yapısıdır.

- enum sabitlerinin varsayılan değeri int türündedir. Fakat diğer tamsayı türleri ile de belirtilebilir. Tanımlanması:
  - enum isim{sabit1,sabit2,…}
  - enum isim : byte{sabit1,sabit2,…}

- Numaralandırma sabitlere sıfırdan başlayarak artan birer tamsayı değeri verilmesi esasına göre gerçekleştirilir.

```
using System;

enum GUNLER : byte
{
    PAZARTESI,
    SALI,
    CARSAMBA,
    PERSEMBE,
    CUMA,
    CUMARTESI,
    PAZAR
}

class Numaralandirma
{
    public static void Main()
    {
        Console.WriteLine((int)GUNLER.PAZARTESI);
        Console.WriteLine((int)GUNLER.PAZAR);
    }
}
```

- enum sabitlerine enumAdı.Sabit şeklinde ulaşılır.
- enum sabitleri de birer veri tipi oldukları için fonksiyona parametre olarak gönderilebilir.
- Tanımlama içinde farklı sayı değerleri vermek için sabitler “=“ ile farklı değerlere atanabilir.
- System.Enum sınıfından türetildikleri için bazı metotlar ile sabit sembollerin isimleri alınabilir.

```
using System;

enum GUNLER : byte
{
    PAZARTESI,
    SALI,
    CARSAMBA,
    PERSEMBE,
    CUMA,
    CUMARTESI,
    PAZAR
}

class Numaralandirma
{
    public static void Main()
    {
        string[] sabitler = Enum.GetNames(typeof(GUNLER));
        foreach (string s in sabitler)
            Console.WriteLine(s);
    }
}
```

ÖRN:

- Belirli bir dersi alan öğrencilerin bilgilerinin bellekte kaydedildiği ve üzerinde bazı işlemlerin yapılabildiği aşağıdaki ekran çıktısı verilen programı yazınız. Örnek Yapı:

```
public struct Ogrenci
{
   public int Numara;
   public string Ad;
   public string Soyad;
   public int Not;
}
```