

4 Çok-Biçimlilik (Polymorphism)

Nesneye dayalı programlamada, bir nesnenin kendi özellikleri dışına çıkıp başka bir nesne gibi davranmasına çok-biçimlilik (polymorphism) denir.

2 tür çok biçimlilik vardır;

- Statik / Derleme Zamanı Çok-biçimlilik (Compile Time)
 - Fonksiyon Aşırı Yükleme (Function Overloading)
 - Yapıcı Aşırı Yükleme (Constructor Overloading)
 - Operator Overloading
- Dinamik / Çalışma Zamanı Çok-biçimlilik (Run Time)
 - Kalıtım (Inheritance)
 - Virtual
 - Abstract
 - Interface Polymorphism
 - Sealed

4.1 Fonksiyon Aşırı Yükleme (Function Overloading)

Aynı sınıf içinde aynı isme sahip birden fazla fonksiyon tanımlamaktır.

Fonksiyonların içerdikleri veri tipleri ya da parametre (argüman) sayıları farklı olmalıdır.

```

public class islem
{
    #region Parametre Sayısının Farklı Olması
    //parametre sayısı farklı
    public int toplama()
    {
        return 0;
    }

    public int toplama(int sayi)
    {
        return sayi;
    }

    public int toplama(int sayi1, int sayi2)
    {
        return sayi1 + sayi2;
    }

    public int toplama(int sayi1, int sayi2, int sayi3)
    {
        return sayi1 + sayi2 + sayi3;
    }
}
#endregion

```

Parametre Türlerinin Farklı Olması

```

public class islem
{
    Parametre Sayısının Farklı Olması

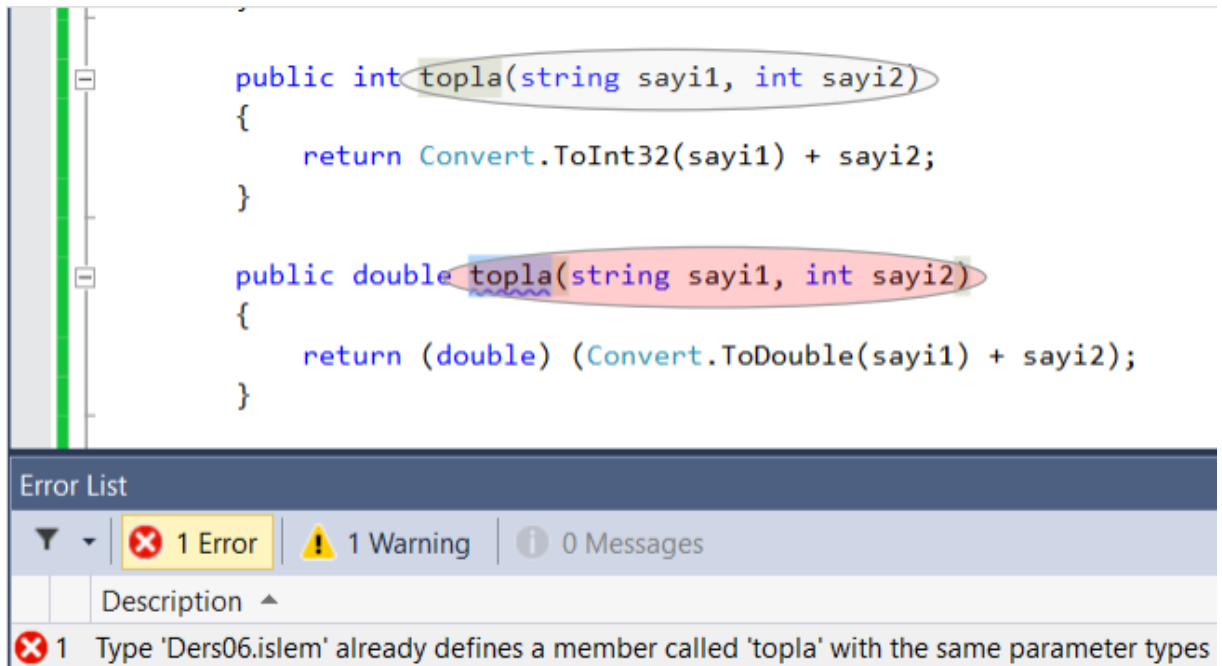
    #region Parametre Türlerinin Farklı Olması
    //parametre türü farklı
    public int toplama(string x)
    {
        return Convert.ToInt32(x);
    }

    public int toplama(string sayi1, int sayi2)
    {
        return Convert.ToInt32(sayi1) + sayi2;
    }

    public int toplama(int[] ss)
    {
        int t = 0;
        for (int i = 0; i < ss.Length; i++)
            t += ss[i];
        return t;
    }
}
#endregion

```

Aynı parametre özellikleri ile farklı geri dönüş tipine sahip fonksiyon tanımlanamaz:



Bir sınıf içinden aynı isme ait fonksiyonlara aşağıdaki şekilde erişilebilir: • Nesne ismini yazıp Parantez açtığınızda 7 function'a dikkat.

```
class Program
{
    static void Main(string[] args)
    {
        islem _i = new islem();

        Console.WriteLine(_i.topla());
        Console.WriteLine(_i.topla(3));
        Console.WriteLine(_i.topla(4,5));
        Console.WriteLine(_i.topla(4,5,6));
        Console.WriteLine(_i.topla("3"));
        Console.WriteLine(_i.topla("5",7));

        int[] sayilar = {55,47,36,14,67};

        Console.WriteLine(_i.topla(sayilar));

        _i.topla()
        ▲ 1 of 7 ▼ int islem.topla()

    }
}
```

4.2 Yapıcı Aşırı Yükleme (Constructor Overloading)

Yapıcılar da fonksiyonlara benzer şekilde aşırı yüklenebilir:

```
class oyuncu
{
    private string ad;
    private string mevki;

    public oyuncu()
    {
        ad = "aaa";
        mevki = "bbb";
    }
    public oyuncu(string adi )
    {
        ad = adi;
        mevki = "bbb";
    }
    public oyuncu(string adi, string mevkisi)
    {
        ad = adi;
        mevki = mevkisi;
    }
}
```

4.3 Operatör Aşırı Yükleme (Operator Overloading)

Tamsayı değişkenleri toplama (+) operatörü toplayıp başka bir integer değişken içine atarız. Sonuç iki değişkenin toplamıdır.

Console.WriteLine içinde + operatörünü string ve integer değerler arasında kullandığımızda ise toplama yapılmaz. Integer'da string gibi davranır ve string birleştirme işlemi yapılmış olur.

Örnek:

```
using System;
namespace operatorAsiriYukleme
{
    class Program
    {
        static void Main(string[] args)
        {
            DateTime basla = new DateTime();
            basla = DateTime.Now;
        }
    }
}
```

```

        Console.WriteLine("Çalışmaya Başlama Zamanı :" +
        basla.ToString()); // + operatörü string birleştirme yapıyor

        for (int i = 0; i < 1000000000; i++) {

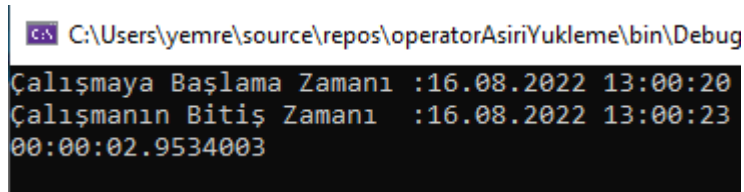
            DateTime bitir = new DateTime();
            bitir = DateTime.Now;
            Console.WriteLine("Çalışmanın Bitiş Zamanı :" + bitir); //
            + operatörü string birleştirme yapıyor

            TimeSpan gecenSure = new TimeSpan();
            gecenSure = bitir - basla;
            Console.WriteLine(gecenSure);

            Console.ReadKey();
        }
    }
}

```

Çıktısı:



```

C:\Users\yemre\source\repos\operatorAsiriYukleme\bin\Debug
Çalışmaya Başlama Zamanı :16.08.2022 13:00:20
Çalışmanın Bitiş Zamanı :16.08.2022 13:00:23
00:00:02.9534003

```

4.3.1 Kişisel Operatör Aşırı Yükleme

İşlemlerde kullandığımız operatörleri direk olarak sınıflarımızın üzerinde kullanamayız:

```

using System;
namespace operatorAsiriYukleme
{
    class ornekNesne
    {
        //...
    }
    class Program
    {
        static void Main(string[] args)
        {
            ornekNesne n1, n2 = new ornekNesne();
            ornekNesne n3 = n1 + n2;
            ...
        }
    }
}

```

Aşağıdaki hatayı alırsınız:

```
C:\Users\yemre\source\repos\operatorAsiriYukleme\Program.cs(14,29,14,36): error CS0019: Operator '+' cannot be applied to operands of type 'ornekNesne' and 'ornekNesne'
```

Bu işlemi operatörlerin aşırı yüklenmesi özelliği ile tasarladığımız sınıflar içinde yazabiliriz. C# ile operatörlerin aşırı yüklenmesi için aşağıdaki metod tanımlarını kullanılır:

```
public static <dönüş-değeri> operator <operatör-işareti>(<parametre>)  
{ }
```

Örnek:

```
using System;  
namespace operatorAsiriYukleme  
{  
    class Sayi  
    {  
        private int deger;  
  
        public Sayi(int gelenDeger)  
        {  
            this.deger = gelenDeger;  
        }  
  
        public void Yazdir()  
        {  
            Console.WriteLine(this.deger);  
        }  
  
        public static Sayi operator +(Sayi a, Sayi b)  
        {  
            return new Sayi(a.deger + b.deger);  
        }  
    }  
    class Program  
    {  
        static void Main(string[] args)  
        {  
            Sayi s1 = new Sayi(3);  
            Sayi s2 = new Sayi(5);  
            Sayi s3 = s1 + s2;  
            s3.Yazdir();  
  
            Console.ReadKey();  
        }  
    }  
}
```

Yukarıdaki örnekte (+) toplama operatörü aşırı yüklenmiştir.

Aşırı yükleme sonucu döndürülecek nesne **Sayi** türündendir ve yeni bir nesnedir.

Burada toplama sonucunu **int** veya başka türden bir değişkene aktarmak istediğimizde dönüş değerini değiştiririz:

```
using System;
namespace operatorAsiriYukleme
{
    class Sayi
    {
        private int deger;

        public Sayi(int gelenDeger)
        {
            this.deger = gelenDeger;
        }

        public static int operator +(Sayi a, Sayi b)
        {
            return (a.deger + b.deger);
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            Sayi s1 = new Sayi(3);
            Sayi s2 = new Sayi(5);
            int s3 = s1 + s2;
            Console.WriteLine(s3);

            Console.ReadKey();
        }
    }
}
```