Metotlar ve Fonksiyonlar

- Nesneye yönelik programlama dillerinde genellikle fonksiyonlar “metot” olarak isimlendirilirler. Metot ve fonksiyon olarak ifade edilecek kavramlar aynı anlamda kullanılacaktır.

- Her çalışan C# programı bir ana fonksiyona (Main) sahiptir. Program çalışmaya bu fonksiyondan başlar.

- Tüm kodların bu fonksiyon içine yazılması çok uygun olmaz.

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. Metot, metodu çağıran ana programa bir takım faydalı işler yapar.

- Metodun iş yapması için kendisini çağıran metottan aldığı bilgilere parametre(argüman), kendisini çağıran fonksiyona döndürdüğü bilgiye ise geri dönüş değeri (return value) denir.

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] <geri dönüş tipi> Metot İsmi (parametreler)

{
        Metot kodları…
}
```

- Erişim belirteçleri (public, private..) metoda nerelerden erişilebileceğini ayarlar. Belirtilmediği durumlarda “private” olarak kabul edilir. Yani sadece tanımlandığı sınıf içinde kullanılabilen bir metot olur.

- Geri dönüş değeri, metodun çağıran fonksiyona gönderdiği veririn türüdür.

- Parametreler ise metodun çalışırken ihtiyaç duyduğu çağıran fonksiyondan gönderilen bilgileridir.

Örn:

```
int Topla(int a, int b)
{
   return a+b;
}
```

C#'ta bir metot kullanılacaksa metodun içinde bulunduğu sınıf türünden bir nesne tanımlanır ve '.' operatörü ile metot çağrılır. Ancak "static" olan tanımlanan metotları çağırmak için bir nesne tanımlamaya gerek yoktur.

Eğer metodun içinde olduğu sınıfın içinden çağrılacaksa metodun sadece adı yazılarak çalıştırılabilir, eğer sınıfın dışından çağrılacaksa "SınıfAdı.Metot()" şeklinde çağrılmaktadır.

```
using System;

class metotlar_1
{
    static int Topla(int a, int b)
    {
        return a + b;
    }

    static void Main()
    {
        int sonuc = Topla(10, 20);
        Console.WriteLine(sonuc);
    }
}
```

- Metotlara parametre gönderirken tür dönüşümü kurallarına uyulmalıdır, uygun türler kullanılmalıdır.

- “return” anahtar sözcüğü ile geri değer döndürürken de türlerin uyumlu olmasına dikkat edilmelidir.

- Eğer bir metot geriye değer döndürmeyecekse “void” olarak tanımlanır.

- Metotlar şu şekillerde kullanılabilir:
  - int t = Topla(4, 5);
  - Console.WriteLine(Topla(4,5));
  - int sonuc = Topla(4,Topla(5,6));

```
using System;

class metotlar_2
{
    static void Main()
    {
        Yaz(20);
        Yaz("String İfade...");
        Yaz(12.34f);
    }

    static void Yaz(object o)
    {
        Console.WriteLine(o.ToString());
       // return;
    }
}
```

Geri dönüş değeri olmayan metotlarda return anahtar sözcüğünün yanında herhangi bir ifade olmadan da kullanılabilir.

Metot Özellikleri

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

- Metot çağırırken parametreler eksiksiz girilmelidir. Eksik parametreli metot çağrıları geçersizdir.

- Metotların geri dönüş değeri herhangi bir türden olabilir. Geri dönüş değeri olmayan metotlar "void" anahtar sözcüğü ile bildirilmelidir.

- Metotların geri dönüş değerini uyumlu bir türden nesneye atamak gerekir. Büyük türden küçük bir türe tür dönüştürme işlemi yapılmadıkça atayamayız. Bilinçsiz tür dönüşümü yapılan türler arasında ise bu sorun yaşanmaz.

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

```
static void islem(int a, int b)
    {
        return a + b;
    }
static void Main(string[] args)
    {
        int c = islem(5, 4);
    }
```

Geri dönüş değerinde yapılan gizli tür dönüşümleri ve tür uyumsuzluk kuralı metot parametreleri için de geçerlidir.
Örn:

```
static void islem(int a, int b)
    {
        Console.WriteLine(a + b);
        return;

    }
    static void Main(string[] args)
    {
        double x, y;
        x = 5.45D;
        y = 6.88D;
        islem(x, y);
        Console.ReadLine();
    }
```

Örn:

```
static void islem(double a, double b)
    {
       Console.WriteLine(a + b);
       return;

    }
    static void Main(string[] args)
    {
       int x, y;
       x = 5;
       y = 6;
       islem(x, y);
       Console.ReadLine();
    }
```

C ve C++ dillerinde bir fonksiyonun parametre almadığını açıkça bildirmek için parametre parantezine void anahtar sözcüğü yazılır, fakat C#'da böyle bir kullanım yoktur.

Bir metot içerisinde başka bir metot 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();
     }

  }
```

- C ve C++ dillerinde bir fonksiyonun geri dönüş değeri belirtilmediği durumlarda varsayılan olarak int kabul edilmekteydi. Oysa C#'da geri dönüş değeri mutlaka belirtilmelidir. Geri dönüş değeri herhangi bir veri türü ya da void olabilir.

- Metodun içinde tanımlanan değişkenler dışında geçersizdir.

- Metot parametre parantezindeki değişkenlerin türlerinin tek tek belirtilmesi gerekir. ',' ile ortak tür bildirimi yapılamaz.

```
static int islem(int a,  b)
    {          return a + b;
    }
static void Main(string[] args)
    {
        int c = islem(5, 4);
    }
```

Yine, metot parametrelerinin metot içerisinde tanımlanması geçersizdir.

```
static int islem(int a,  b)
    {       int b;
            return a + b;
    }
static void Main(string[] args)
    {
        int c = islem(5, 4);
    }
```

- 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.

```
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);
    }
}
```

```
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);
    }
}
```

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

Metot Parametresi Olarak Diziler

Diziler metotlara parametre olarak aşağıdaki gibi verilir:

```
static void DiziYaz (int [ ] dizi)
{   ……
    ……
    ……
}
```

```
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);
    }
}
```

Değer ve Referans Parametreleri

- C++ dilinde bir nesnenin değerini metoda aktarmaya değer ile parametre geçme (call by value), bir nesnenin referansını metoda geçmeye ise referans ile parametre aktarımı (call by reference) denilmektedir.

- Değer ve Referans türdeki nesneler metotlara parametre olarak gönderilirken dikkat edilmesi gereken noktalar vardır.

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

- Referans tiplerde ise durum farklıdır. Referans parametre ile metot içinde işlem yapmak nesnenin değerini değiştirebilir.

- Normal koşullarda parametre aktarımlarında değer tipleri ver referans tipler kendi türlerine fonksiyona aktarılırlar. Bazı durumlarda değer tiplerinin metoda referans tip olarak aktarılması istenebilir. Bunun için “ref” ve “out” anahtar sözcükleri kullanılır.

```
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.

```
using System;

class metotlar_5b
{
    static void ReferansTipAktarim(int[] Sayi)
    {
        Sayi[0] = 30;
    }

    static void Main()
    {
        int[] x = { 100, 200 };
        Console.WriteLine(x[0]);

        ReferansTipAktarim(x);
        Console.WriteLine(x[0]);
    }
}
```

Diziler, referans türleri oldukları için metotlara referans olarak geçirilirler, kopyalanan sadece dizinin bellekteki yerini gösteren referanstır.

Ref ve Out Anahtar Sözcükleri

- 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.

```
using System;

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

    static void Main()
    {
        int x = 25;

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

ref parametreleri aktarılırken mutlaka ilk değer verilmelidir. Eğer ki, ilk değer verilmeyecekse ref yerine out anahtar sözcüğü kullanılmalıdır.

```
using System;

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

    static void Main()
    {
        int x = 25;

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

```
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)

- Metotları aşırı yükleyerek (overloading) aynı isimli birden fazla metot tanımlanabilir.

Çalışma zamanında bu metotlar nasıl belirlenecek?

İsimler aynı olduğuna göre hangi metodu çağırdığımız nasıl anlaşılacak?

- 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.

```
using System;
class AsiriYukleme
{
    static void OrnekMetot()
    {
        Console.WriteLine("1.Metot");
    }
    static void OrnekMetot(int x, int y)
    {
        Console.WriteLine("2.Metot");
    }
    static void OrnekMetot(string a, string b)
    {
        Console.WriteLine("3.Metot");
    }
    static void OrnekMetot(int x)
    {
        Console.WriteLine("4.Metot");
    }
    static void OrnekMetot(float x,float y)
    {
        Console.WriteLine("5.Metot");
    }

    static void Main()
    {
        OrnekMetot(56);
        OrnekMetot("Metot", "Yukleme");
        OrnekMetot();
        OrnekMetot(74, 56);
        OrnekMetot(12, 34f);
        OrnekMetot('a', 'e');
    }
}
```

- 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();
}
```

- Aynı parametrelere sahip geri dönüş değerleri farklı olan fonksiyonlar derleme anında hataya neden olurlar. Çünkü metotların geri dönüş değerleri ayırt edici bir özellik değildir.

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.

```
using System;

class DegiskenParametre
{
    static int Topla(params int[] sayilar)
    {
        if (sayilar.Length == 0) return 0;

        int t = 0;

        foreach (int x in sayilar)
            t += x;
        return t;
    }

    static void Main()
    {
        Console.WriteLine(Topla());
        Console.WriteLine(Topla(5,9,150));
        Console.WriteLine(Topla(19,80));
        Console.WriteLine(Topla(2,9,25,65,98,27));
    }
}
```

- Değişken sayıda parametre içeren metotlar, aşırı yüklenmiş metotlar olduğunda değerlendirmeye alınmazlar.

```
using System;

class DegiskenParametre2
{
    static void Yaz(object o)
    {
        Console.WriteLine("1.Metot:" + o.ToString());
    }

    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()
    {
        Yaz(25);
        Yaz("Deneme1", "Deneme2");
        Yaz('a');
        Yaz('z',1, 23f, 4, 56, "abc");
    }
}
```

Özyineli (Recursive) Metotlar

- Kendi kendini çağıran metotlara özyineli(recursive) metot denilir. Bu metotlarda, çağrımı sonlandıran bir kontrol yapısı da olmalıdır.

```
using System;

class Ozyineleme
{
    static int Faktoriyel(int f)
    {
        if (f == 0) return 1;
        return f * Faktoriyel(f - 1);
    }

    static void Main()
    {
        Console.WriteLine(Faktoriyel(5));
        Console.WriteLine(Faktoriyel(0));
        Console.WriteLine(Faktoriyel(15));
    }
}
```

```
using System;

class Ozyineleme2
{
    static void BitYaz(int b)
    {
        if (b == 0) return;

        BitYaz(b >> 1);
        Console.Write(b & 1);
    }

    static void Main()
    {
        BitYaz(65536); Console.WriteLine();
        BitYaz(756); Console.WriteLine();
    }
}
```

Main Metodu

- Main metodunun yapı olarak diğer metotlardan farkı yoktur. Programların buradan çalışmaya başlaması diğer metotlardan ayıran özelliğidir.

- Main metodunun int türünden bir geri dönüş değeri de olabilir. Bu değer işletim sistemine programın çalışması hakkında bilgi vermek amacıyla kullanılır.

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

  – static int Main(string[] args)

```
using System;

class KomutSatirParametreleri
{
    static void Main(string[] args)
    {
        for (int i = 0; i < args.Length; i++)
            Console.WriteLine("{0}. Parametrenin Değeri = {1}",
                              i, args[i]);
    }
}
```

System.Math () Sınıfı

- Belirli matematiksel işlemleri yapmak için System isim alanında bulunan Math sınıfı kullanılır. Math sınıfının metotları static olduğu için bir nesne tanımlaması yapmadan aşağıdaki gibi kullanabiliriz:

  – Math.MetotIsmi(parametreler);

Math sınıfının 2 temel özelliği vardır. Örn:

```
static void Main(string[] args)
        {
            double e = Math.E;
            double pi = Math.PI;
            Console.WriteLine("e={0,5} ****** pi={1,5}",
              e, pi);
            Console.ReadLine();

        }
```

Abs(x)	Bir sayının mutlak değerini alır.
Cos(x)	Bir sayının kosinüsünü alır.
Sin(x)	Bir sayının sinüsünü alır.
Tan(x)	Bir sayının tanjantını alır.
Ceiling(x)	x sayısını x'ten büyük en küçük tamsayıya yuvarlar.
Floor(x)	x sayısını x'ten küçük en büyük tamsayıya yuvarlar.
Max(x,y)	x ve y sayısından en büyüğünü bulur.
Min (x,y)	x ve y sayısından en küçüğünü bulur.
Pow(x,y)	x^y sayısını hesaplar.
Sqrt(x)	$x^{1/2}$ sayısını hesaplar.
Log(x)	lnx sayısını hesaplar.
Exp(x)	e^x sayısını hesaplar.
Log10(x)	logx sayısını hesaplar.