Bölüm 12

Metotlar

Metot Nedir?

Her bilgisayar dili değişken ve fonksiyon kavramlarına sahiptir. Bu tesadüfen değil, işin doğasından kaynaklanan bir durumdur. Değişkenler, bilgisayar programının işleyeceği ham maddedir. Bu ham maddelerin işlenip ürün haline getirilmesi için kullanılan araçlar fonksiyonlardır.

Nesne Yönelimli (Object Oriented- OO) Programlama Dillerinde her şey sınıflar (`class`) içinde tanımlanır. Bir sınıfa ait değişkenler o sınıfın özelliklerini ortaya koyar. O nedenle, sınıf değişkenlerine, çoğunlukla, özelik (`property`) denilir. Benzer olarak bir sınıf içinde tanımlı fonksiyonlar o sınıfın tavır, davranış ve eylemlerini ortaya koyar. Bu ayrımı gözetmek için olsa gerek, OO Programlama dillerinde fonksiyonlara `metot` denir. Bundan sonra, biz de bu geleneğe uyarak, bir sınıfta tanımlanan değişken yerine `özelik`, fonksiyon yerine `metot` diyeceğiz. Ancak, özellikle alışkanlıklarımıza dayanan vurgu gerektiğinde değişken ve fonksiyon adlarını kullanmaktan da çekinmeyeceğiz. Okurun, `değişken ile özeliğin` ve `fonksiyon ile metodun` eş anlamlı olduğunu daima anımsayacağını umuyoruz. Değişkenlere neden başka ad verdiğimizi birazdan anlamış olacağız.

Fonksiyon kavramına geçmeden önce, uzaylı dostumuzun bir uyarısı daha var. Onu dinlesek iyi olur.

Anadilimizi öğrenirken bile sistematik bir yöntem izlemedik. Bazı kalıpları (gramer) çevremizden duyduğumuz gibi kullanmaya başladık. Ana dilimizi kullanmayı öğrendikten çok sonra onun gramerini öğrendik. Programlama dillerinin öğrenimi de çoğunlukla ona benzer. Başlangıçta bir çok sözdizimini (syntax), nedenini sormadan,

olduğu gibi kullanırız. Sonra onun nedenini algılamamız daha kolay olur.

Static Öğe

Şimdiye dek bir çok kez kullandığımız `Main()` metodunu `static` anahtar sözcüğü ile niteledik. Başka metot ve değişkenler (özelik) için de aynı nitelemeyi kullandık. Bir sınıfta tanımlı bir değişken veya metodu `static` diye nitelemenin OO programlamada özel bir işlevi vardır. Genel kural olarak, bir sınıfta tanımlı değişkenler ve fonksiyonlar, ancak o sınıfa ait bir nesne (object) yaratıldıktan sonra o nesne içinde işlevsellik kazanırlar. Bunun istisnai hali değişkenin veya metodun önüne `static` nitelemesi konulduğunda oluşur. Başka bir deyişle şunu söyleyebiliriz:

İpucu

`static` sözcüğü ile nitelenen değişken veya metot, ait olduğu bir nesne (object) yaratılmadan kullanılabilir.

`static` sözcüğünün işlevini sınıf konusunu işlerken daha ayrıntılı açıklayacağımızı, ama o zamana kadar onu gerektiği yerde kullanmak zorunda olduğumuzu unutmayınız.

Metot Kavramı

`Metot` kavramı, matematik derslerinden bildiğimiz fonksiyon tanımından başka bir şey değildir. Matematikte

```
f : X --> Y  , {x -->y}
```

simgelerinin ifade ettiği anlam şudur: Fonksiyonun adı `f` dir. `f` fonksiyonu tanım kümesi denilen `X` kümesinden, değer kümesi denilen `Y` kümesi içine tanımlıdır. `f` fonksiyonu `X` kümesinden aldığı her `x` öğesini, fonksiyon kuralı denilen bir dönüşüm yöntemiyle `Y` kümesi içindeki `y` öğesine götürür (eşler). `X` öğesine fonksiyonun değişkeni, `y` öğesine `x` öğesinin `f` altındaki dönüşümü (resmi) diyoruz. Her `x` değişkenine bir ve yalnız bir `y` öğesi karşılık gelir. Başka bir deyişle, bir değişken birbirinden farklı iki öğeye eşlenemez. Buradan şunu hemen anlıyoruz. Her fonksiyonun bir tanım kümesi `( X )`, bir değer kümesi `( Y )` ve bir dönüşüm kuralı `{x --> y}` vardır.

Matematikten anımsadığımız bu kavramı, şimdi, bilgisayar diliyle ifade edelim. Açıklamayı basit bir örnek üzerinde yaparsak konu daha kolay anlaşılacaktır. Aşağıdaki fonksiyon bir tamsayının karesini hesaplar.

```
int KaresiniBul(short x)
{
    int y ;
    y = x * x ;
    return y ;
}
```

Bu tanımda

```
int KaresiniBul(short x)
```

satırına `fonksiyonun başlığı` diyeceğiz. Fonksiyon başlığı, fonksiyon hakkında bize gerekli başlıca bilgileri verir. Bu nedenle, bazı kaynaklar ona fonksiyonun mührü (signature) derler. Biz fonksiyon başlığı terimini tercih edeceğiz. Önce başlığın bize verdiği bilgilere bakalım:

Fonksiyonun adı `'KaresiniBul'` bitişik sözcükleridir.

Fonksiyonun adının sağına yazılan `(short x)` ifadesi, fonksiyonun tanım kümesinin `short` veri tipi olduğunu ve değişkeninin `x` simgesiyle gösterildiğini belirtir.

Fonksiyonun adının soluna yazılan `int` veri tipi fonksiyonun değer bölgesidir.

Başlıktaki satırdan sonra gelen { } bloku fonksiyona işlevini yaptıran deyimleri içeren bloktur. Bu bloku `fonksiyon bloku` veya `fonksiyon gövdesi` diye adlandıracağız.

{ } gövdesi içinde `y = x*x` çarpma ve atama işlemleri fonksiyonun yapacağı iştir. `x*x` çarpımının sonucu `y` değişkenine atanmış ve `y` değeri fonksiyon-değeri olarak belirlenmiştir. *[x öğesinin eşleneceği y öğesini belirleyen fonksiyon kuralı]*. Fonksiyonun alacağı değeri `return` anahtar sözcüğü ile belirtiyoruz:

```
return y;
```

Parametreler ve yerel değişkenler

Bilgisayar programlama dillerinde, fonksiyonun başlığındaki değişkene `parametre` denilir. Bunun nedenini açıklamak için, yukarıdaki { } gövdesine bakalım.

Fonksiyonun { } gövdesinde `int` tipinden bir `y` değişkeni tanımlanmıştır. Dolayısıyla bu fonksiyon iki farklı nitelikte değişken kullanmaktadır. Başlıktaki `(short x)` değişkeni ve fonksiyon gövdesi içindeki `int y` değişkeni.

Bu iki değişkeni birbirlerinden ayırt etmek için,

fonksiyon başlığındaki x değişkenine `parametre` ve

{ } gövdesi içindeki `y` değişkenine fonksiyonun `yerel değişkeni` denilir.

İpucu

Bilgisayar programlarında fonksiyona, matematikteki gibi tek harfli ad takmak yerine, genellikle, onun işlevini ima eden sözcükler takılır. Kaynak programı okuyan kişinin fonksiyonun işlevini hemen kavraması için bu iyi bir gelenektir. Bunu alışkanlık edininiz. Ancak, derleyici açısından fonksiyona takılan ad hiç önemli değildir. Tek harfli ad olsa da birden çok sözcükten oluşan uzun ad verilse de sistem ona işlevini eksiksiz gördürecektir.

Bilgisayar programlarında fonksiyonların parametrelerinin her biri bir veri tipi üzerinde tanımlıdır. Bu veri tip(ler)i onun tanım kümesidir. Her fonksiyon parametreleri ve yerel değişkenleri üzerinde, önceden belirlenen bazı işlemleri yapar ve sonunda bir değer verir (dönüşüm). Bu değer bir veri tipi içindedir. O veri tipi, fonksiyonun değer kümesidir.

C# dilinde her şeyin bir sınıf içinde tanımlanması gerektiğini söylemiştik. Öyleyse, yukarıdaki fonksiyon tanımını bir sınıf içine taşımalıyız. Fonksiyonumuzu istediğimiz bir sınıf içinde tanımlayabileceğimizi ileride göreceğiz. Ama, şimdi işleri kolaylaştırmak için onu, `Main()` fonksiyonunu da içeren bir sınıf içine taşıyalım.

Metot01.cs

```
using System;

namespace Metotlar
{
    class Metot01
    {
        static int n;

        static void Main(string[] args)
        {
            n = KaresiniBul(9);
            Console.WriteLine(n);
        }

        static int KaresiniBul(short x)
        {
            int y;
            y = x * x;
            return y;
        }
    }
}
```

Main() metodu

C# dilinde bir programın bütün işlevini yaptıran ana metottur. Programın kullanacağı bütün deyimleri tek başına içerebileceği gibi, modüler yapıda yazılan programlarda öteki sınıflarda tanımlanan metotları ve değişkenleri çağırır ve belirlenen sırayla işleme sokar. Her programda ana metot olma işlevini üstlenen bir ve yalnızca bir tane `Main()` metodu vardır. Bir programda başka Main metotları olsa bile, onlar bu programda ana metot olma sorumluluğunu taşıyamazlar.

`Main()` bir metot olduğuna göre, tanımı genel sözdizimi kurallarına uyar. Çoğunlukla başlığını

```
static void Main(string[] args)
```

biçiminde yazarız. Bunun nedenini ilerleyen bölümlerde daha iyi anlayacaksınız. Şimdilik şunları söylemekle yetinelim: Yukarıdaki `Main()` metodunun,

`args` adını taşıyan parametresi `string` tipi bir array'dir.

Değer kümesi `void` kümesidir. Bu küme matematikteki `boş küme` gibidir. Hiç bir öğesi yoktur. Dolayısıyla, bu örnekte `Main()` metodu bir değer almıyor, yalnızca başka bir metodu çağırıp onun işlevini yapmasını sağlıyor.

Başlıktaki `static` anahtar sözcüğünü mutlaka kullanmalıyız. Onun ne iş yaptığını ileride göreceğiz.

Yukarıdaki örnekte, `Metot01` adlı sınıf içinde,

```
static int n;
```

değişkeni tanımlıdır. Sınıfa ait değişkenlere özelik diyeceğimizi söylemiştik. Dolayısıyla, `n` değişkeni `Metot01` sınıfının bir özeliğidir. Ayrıca bu sınıf içinde `Main()` metodu ile `KaresiniBul()` metodu tanımlıdır. `Main()` metodunun gövdesinde yazılan

```
KaresiniBul(9);
```

deyimine, *fonksiyonu parametre değeriyle çağırmak* denir. Fonksiyon çağrılırken onun `x` parametre adı yerine, o parametreye verilen 9 değeri ile çağrılmaktadır. Başka bir deyişle, fonksiyon çağrılırken `x*x` soyut işlemi yerine `9*9` somut işlemini yapması istenmektedir. Bu olgu, matematikte `f(x) = x^2` fonksiyonu tanımlı iken `f(9) = 9^2 = 81` işlemini yapmak gibidir.

`Metot02` sınıfı içinde tanımlanan `n` değişkeni, `KaresiniBul()` metodu ve `Main()` metodu `static` anahtar sözcüğü ile nitelenmişlerdir. Bunun nedenini bu bölümün başında açıklamıştık. Ayrıca, anımsayacaksınız, bu bölümün başında sınıf içinde tanımlanan değişkenlere '`özelik`' diyeceğiz demiştik.

İpucu

Böylece, bir sınıf içinde olabilecek üç farklı nitelikteki değişkeni ayrı ayrı gruplamış oluyoruz:

Fonksiyonun yerel değişkeni,

fonksiyonun parametresi ve

sınıfın özeliği.

Yukarıdaki basit örnekte görüldüğü gibi, bu değişken gruplarının kullanılışları ve işlevleri birbirlerinden farklıdır.

Ayrıca bir fonksiyonun hiç parametresi veya yerel değişkeni olmayabileceği gibi, birden çok parametresi veya yerel değişkeni olabilir.

Benzer olarak, bir sınıfın hiç özeliği veya metodu olmayabileceği gibi, birden çok özeliği ve metodu olabilir. Bunları örnekler üzerinde göreceğiz.

Aşağıdaki program bir metin içindeki küçük harf, büyük harf, rakam ve `alfasayısal` (harf veya rakam) olmayan karakterlerin sayılarını bulur. Her bir sayımı yapmak için, `KarakterleriSay` sınıfı içinde dört ayrı metot tanımlanmıştır. Metotların `Main()` tarafından çağrılabilmesi için herbiri `static` nitelemesini almıştır.

KarakterleriSay.cs

```
using System;

namespace Metotlar
{
    class KarakterleriSay
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Bir satır yazıp Enter'e basınız: ");
            string s = Console.ReadLine();
            Console.WriteLine("Küçük Harf Sayısı : " + KüçükHarfSay(s));
```

```
            Console.WriteLine("Büyük Harf Sayısı : " + BüyükHarfSay(s));
            Console.WriteLine("Rakam Sayısı : " + RakamSay(s));
            Console.WriteLine("Alfasayısal Olmayan Karakter Sayısı :" +
AlfasayısalDeğilseSay(s));
        }


        static int KüçükHarfSay(string str)
        {
            int küçükHarfSayısı = 0;
            foreach (char harf in str)
                if (Char.IsLower(harf)) küçükHarfSayısı++;
            return küçükHarfSayısı;
        }

        static int BüyükHarfSay(string str)
        {
            int büyükHarfSayısı = 0;
            foreach (char harf in str)
                if (Char.IsUpper(harf)) büyükHarfSayısı++;
            return büyükHarfSayısı;
        }

        static int RakamSay(string str)
        {
            int rakamSayısı = 0;
            foreach (char harf in str)
                if (Char.IsDigit(harf)) rakamSayısı++;
            return rakamSayısı;
        }

        static int AlfasayısalDeğilseSay(string str)
        {
            int alfaSayısalDeğilSayısı = 0;
            foreach (char harf in str)
                if (!Char.IsLetterOrDigit(harf)) alfaSayısalDeğilSayısı+
+;
            return alfaSayısalDeğilSayısı;
        }


    }
}
```

Çıktı

```
Bir satır yazıp Enter'e basınız:
C# dersine başlayalı 3 gün oldu. Şimdi program yazabiliyorum.
Küçük Harf Sayısı : 47
Büyük Harf Sayısı : 2
Rakam Sayısı : 1
Alfasayısal Olmayan Karakter Sayısı :11
Devam etmek için bir tuşa basın . . .
```

`static` nitelemesinin işlevini daha iyi açıklamak için Birkaç örnek program daha inceleyeceğiz. Aşağıdaki programı derlemeyi deneyiniz.

Tarih01.cs

```
using System;

namespace Metotlar
{
    class Tarih
    {
            static void Main(string[] args)
            {
                TarihYaz();
            }


            void TarihYaz()
            {
                DateTime gününTarihi = DateTime.Now;
                Console.WriteLine(gününTarihi.ToString("dd/MM/yyyy"));
            }
        }
    }
```

Derleyici size şu hata mesajını iletecektir:

```
Error   1   An object reference is required for the non-static field, method, or property 'Metotlar.Tarih.TarihYaz()'
    C:\vsProjects\Metotlar\Metotlar\Metot01.cs   9   17   Metotlar
```

Bu iletinin bize söylediği şey şudur: 9-uncu satırdaki `static` olmayan `TarihYaz()` metodu ancak bir nesne içinde kullanılabilir.

Şimdi bir nesne yaratmadan, `static` nitelemesiyle derleyicinin itirazını kaldırabileceğimizi, aşağıdaki program gösterecektir. Bunun için `TarihYaz()` metodunu `static` olarak nitelemek yetecektir.

Tarih02.cs

```
using System;

namespace Metotlar
{
    class Tarih
    {
            static void Main(string[] args)
            {
                TarihYaz();
            }


            static void TarihYaz()
            {
                DateTime gününTarihi = DateTime.Now;
                Console.WriteLine(gününTarihi.ToString("dd/MM/yyyy"));
            }
        }
```

```
    }
```

Çıktı

```
10/08/2008
```

Nesne yaratma

Şimdi de `static` nitelemesini kullanmamak için `Tarih` sınıfından bir nesne yaratalım. Sınıfları incelerken bu konuyu daha ayrıntılı ele almıştık. Burada şu kadarını tekrar anımsayalım: `Tarih` sınıfı tanımlı iken, `Main()` metodu içinde

```
Tarih t = new Tarih();
```

deyimi `Tarih` sınıfının `t` adlı bir nesnesini yaratır. Bu nesne içindeki `TarihYaz()` metodunu çağırmak (nesne içindeki metoda erişmek) için

```
t.TarihYaz();
```

deyimi yeterlidir. Aşağıdaki programı derleyip koşturunuz; önceki program ile aynı çıktıyı verdiğini göreceksiniz.

Tarih03.cs

```
using System;

namespace Metotlar
{
    class Tarih
    {
            static void Main(string[] args)
            {
                Tarih t = new Tarih();
                t.TarihYaz();
            }


            void TarihYaz()
            {
                DateTime gününTarihi = DateTime.Now;
                Console.WriteLine(gününTarihi.ToString("dd/MM/yyyy"));
            }
        }
    }
```

Çıktı

```
10.07.2008

Devam etmek için bir tuşa basın . . .
```

Hazır metotlar

C# derleyicisi çok zengin bir kütüphaneye sahiptir. Bu kütüphanede çok sayıda namespace, her namespace içinde sınıflar ve sınıflar içinde pek çok işi görecek metotlar vardır. Buna rağmen, kullanıcı gerekseme duyduğunda kendi namespace'lerini, sınıflarını ve metotlarını serbestçe yaratabilmektedir. Yukarıda yaptıklarımız, kullanıcının yaratabileceği sınıf ve metotların basit örnekleridir.

Şimdiye kadar kullandığımız `Write()`, `WriteLine`, `Read()`, `ReadLine()` metotları `Console` sınıfına aittir. `isLower()`, `isUpper()`, `isDigit()`, `IsLetterOrDigit()` metotları `Char` sınıfına aittir. İlerleyen bölümlerde C# kütüphanesindeki başka sınıf ve metotları kullanmayı öğreneceğiz. Bunlar hakkında daha ayrıntılı bilgiyi `msdn` web sitesinden her zaman öğrenebilirsiniz. Ama başlangıçta kütüphaneyi ezberlemeye kalkmayın. Sizden beklenen, gerektiğinde o kütüphaneyi nasıl kullanacağınızı ve aradığınızı nasıl bulacağınızı öğrenmenizdir.

System.Math sınıfı

Matematik fonksionları her dilde çok önem taşır. Onlar olmadan program yazmak hemen hemen olanaksızdır. C# dili matematik metotlarını `System.Math` sınıfı içinde toplamıştır. Math sınıfı mühürlüdür (sealed). Dolayısıyla, ileride göreceğimiz gibi, Math sınıfının kalıtımı (inherited) olamaz. Math sınıfının bütün öğeleri static'dir. O nedenle, Math sınıfının bir nesnesi (object) yaratılamaz. Böyle olduğu için, Math sınıfının metotlarını Math.Sqrt() biçiminde sınıf adıyla birlikte kullanacağız.

Aşağıdaki program o metotları ve çağrılma yöntemlerini göstermektedir.

MathMetotları.cs

```
using System;

namespace Metotlar
{
    public class MathMetotları
    {
        static void Main(string[] args)
        {
            double x,y;
            string sx, sy ;
            Console.WriteLine("Lütfen birinci sayıyı giriniz : ");
            sx = Console.ReadLine();
            x = Double.Parse(sx);

            Console.WriteLine("Lütfen ikinci sayıyı giriniz : ");
            sy = Console.ReadLine();
            y = Double.Parse(sy);

            Console.WriteLine();

            Console.WriteLine("Başlıca Math Metotları :");
            Console.WriteLine("Abs({0}) = {1}", x, Math.Abs (x));
            Console.WriteLine("Ceiling({0}) = {1}", x, Math.Ceiling(x));
            Console.WriteLine("Exp({0}) = {1}", x, Math.Exp(x));
            Console.WriteLine("Floor({0}) = {1}", x, Math.Floor(x));
            Console.WriteLine("IEEERemainder({0},{1}) = {2}", x, y,
Math.IEEERemainder(x,y));
```

```
            Console.WriteLine();
            Console.WriteLine("Log({0}) = {1}", x, Math.Log(x));
            Console.WriteLine("Log10({0}) = {1}", x, Math.Log10(x));
            Console.WriteLine("Max({0}) = {1}", x, Math.Max(x, y));
            Console.WriteLine("Min({0},{1}) = {2}", x, y, Math.Min(x,y));
            Console.WriteLine("Pow({0})  = {1}", x, Math.Pow(x, y));
            Console.WriteLine("Round({0}) = {1}", x, Math.Round(x));
            Console.WriteLine("Sign({0})  = {1}", x, Math.Sign(x));
            Console.WriteLine("Sqrt({0})  = {1}", x, Math.Sqrt(x));
            Console.WriteLine();
            Console.WriteLine("Cos({0})   = {1}", x, Math.Cos(x));
            Console.WriteLine("Sin({0})   = {1}", x, Math.Sin(x));
            Console.WriteLine("TAn({0})   = {1}", x, Math.Tan(x));
            Console.WriteLine("Cotan({0}) = {1} ", x, 1 / Math.Tan(x));
            Console.WriteLine();
            Console.WriteLine("Acos({0})  = {1}", x, Math.Acos(x));
            Console.WriteLine("Asin({0})  = {1} ", x, Math.Asin(x));
            Console.WriteLine("Atan({0})  = {1} ", x, Math.Atan(x));
            Console.WriteLine("Atan2({0},{1}) = {2}", x,y,
Math.Atan2(x,y));
            Console.WriteLine();
            Console.WriteLine("Cosh({0})  = {1} ", x, Math.Cosh(x));
            Console.WriteLine("Sinh({0})  = {1} ", x, Math.Sinh(x));
            Console.WriteLine("TAnh({0})  = {1} ", x, Math.Tanh(x));
            Console.WriteLine("Cotanh({0}) = {1} ", x, 1 / Math.Tanh(x));
        }
    }
}
```

Çıktı

```
Lütfen birinci sayıyı giriniz :
456
Lütfen ikinci sayıyı giriniz :
123
```

Başlıca Math Metotlarının kullanılışı:

Abs(456)	= 456
Ceiling(456)	= 456
Exp(456)	= 1,09215366597392E+198
Floor(456)	= 456
IEEERemainder(456,123)	= -36

Log(456)	= 6,12249280951439
Log10(456)	= 2,65896484266443
Max(456)	= 456
Min(456,123)	= 123
Pow(456)	= Infinity
Round(456)	= 456
Sign(456)	= 1
Sqrt(456)	= 21,3541565040626
Cos(456)	= -0,891991243357829
Sin(456)	= -0,45205267588297
TAn(456)	= 0,506790486172548
Cotan(456)	= 1,97320199823074
Acos(456)	= NaN
Asin(456)	= NaN
Atan(456)	= 1,56860334785422
Atan2(456,123)	= 1,30732978575998
Cosh(456)	= 5,4607683298696E+197
Sinh(456)	= 5,4607683298696E+197
TAnh(456)	= 1
Cotanh(456)	= 1

Özyineli (Recursive) Metotlar

Recursive (özyineli) metot kendi kendisini çağıran metottur. Bilgisayar programcılığında zor problemleri çözen önemli bir algoritmadır.

Fibonacci sayıları

Fibonacci sayıları doğanın çoğalma formülü diye bilinir. Biyolojide, tıpta ve mühendislikte önemli uygulamaları vardır. *Fibonacci* sayılarının ilk 10 tanesini hesaplayan aşağıdaki program recursive bir metot kullanmaktadır.

FibonacciSayıları.cs

```
using System;

namespace Methods
{
    class FibonacciSayıları
    {
        static void Main(string[] args)
        {
            for(int i=1; i<11;i++)
            Console.Write(Fibonacci(i) + "   ");
            Console.WriteLine();
        }

        static int Fibonacci(int n)
        {
            if (n < 2)
                return n;
            else
                return Fibonacci(n - 1) + Fibonacci(n - 2);
        }
    }
}
```

Çıktı

```
1  1  2  3  5  8  13  21  34  55
Devam etmek için bir tuşa basın . . .
```

Recursive metotlar istenen işi yapmak için kendi kendilerini defalarca çağırırlar. Bunu daha iyi algılamak için, yukarıdaki `Fibonacci()` metodunun kaç kez çağrıldığını saydıralım. Bunun için bir sayaç değişkeni tanımlayıp her adımda 1 artmasını sağlamak yetecektir. Bunu yapan aşağıdaki program aynı zamanda Fibonacci sayılarının ne kadar hızla büyüdüğünü göstermektedir. Bunu canlıların çoğalmasına, örneğin bakterilerin çoğalmasına uygularsak Fibonacci sayılarının neden önemli olduğunu anlarız.

FibonacciSayıları02.cs

```
using System;

namespace Methods
{
    class FibonacciSayıları
```

```
    {
        static long sayaç = 0;
        static void Main(string[] args)
        {
            for (int i = 1; i <= 45; i++)
                Console.WriteLine("Fibonacci({0}) = {1:###,###,###,##0}
[{2:###,###,###,###} adım atıdı]", i, Fibonacci(i), sayaç);
        }

        static int Fibonacci(int n)
        {
            sayaç++;
            if (n < 2)
                return n;
            else
                return Fibonacci(n - 1) + Fibonacci(n - 2);
        }
    }
}
```

Çıktı

Fibonacci(n)	**Değeri**	**Kaç Adım**
Fibonacci(1)	= 1	[1 adım atıdı]
Fibonacci(2)	= 1	[4 adım atıdı]
Fibonacci(3)	= 2	[9 adım atıdı]
Fibonacci(4)	= 3	[18 adım atıdı]
Fibonacci(5)	= 5	[33 adım atıdı]
Fibonacci(10)	= 55	[452 adım atıdı]
Fibonacci(15)	= 610	[5.149 adım atıdı]
Fibonacci(20)	= 6.765	[57.290 adım atıdı]
Fibonacci(25)	= 75.025	[635.593 adım atıdı]
Fibonacci(30)	= 832.040	[7.049.122 adım atıdı]
Fibonacci(35)	= 9.227.465	[78.176.299 adım atıdı]
Fibonacci(40)	= 102.334.155	[866.988.830 adım atıdı]
Fibonacci(45)	= 1.134.903.170	[9.615.053.903 adım atıdı]

Faktöriyel hesabını da *recursive (özyineli)* metot ile yapabiliriz.

Aşağıdaki iki metodu inceleyiniz. Birincisi `n!` Sayısını döngü ile, ikincisi özyineli yöntemle buluyor.

```
        int faktoriyel = 1;

            for (int i = n; i >= 1; i--)
                faktoriyel *= i;

        private static long Faktoriyel(int n)
        {
            if (n <= 1)
                return 1;
            else
                return n * Faktoriyel(n - 1);
        }
```

```
using System;

namespace Metotlar
{
    public class Program01
    {
        static void Main(string[] args)
        {
            Console.WriteLine(WhileContinue());
            WhileBreak();
            WhileGoto();
        }
        static int WhileContinue()
        {
            int a = 0;

            while (a < 10)
            {
                a++;
                if (a == 5)
                {
                    a++;
                    continue;
                }
            }
            return a;

        }
        static void WhileBreak()
        {
            int a = 0;

            while (a < 10)
```

```
            {
                a++;
                if (a == 5)

                    break;
            }
            Console.WriteLine(a);
        }

        static void WhileGoto()
        {
            int a = 0;

            while (a < 40)
            {
                if (a == 27)
                    goto sil;
                a++;
            }
        sil:

            Console.WriteLine(a);
        }
    }
}
```

Çıktı

```
10

5

27
```

Kullanıcının Tanımladığı Metotlar

Kendi metodunuzu yaratmanız gerekebilir.

ToplamBul.cs

```
using System;

namespace Methods
{
    class ToplamBul
    {
        static double a = 12.34;
        static double b = 56.78;
        static void Main(string[] args)
        {
            Console.WriteLine(Topla(a,b));
        }

        static double Topla(double x, double y)
        {
            return x+y ;
        }
    }
```

```
}
```

Çıktı

```
69,12
```

Program yazarken, derleyici kütüphanesindeki hazır fonksiyonları kullanmamaya alışmak iyi bir alışkanlıktır. Onlar program yazmayı kolaylaştırdıkları gibi, daima doğru işlem yaptıklarına da güvenebiliriz. Ama zorunluluk olduğunda kendi fonksiyonlarımızı yaratabiliriz.

Aşağıdaki fonksiyon, int tipinden sayıların karekökünü hesaplamak için yazılmıştır. Karekökün tamsayı kısmını doğru hesaplar. Aynı sonucu `Math.Sqrt()` metoduna yaptırmak çok daha kolay ve güvenlidir.

KareKök.cs

```
using System;

namespace Methods
{
    class KareKök
    {
        static int n = 1234;
        static void Main(string[] args)
        {
            Console.WriteLine(Isqrt(n));
            Console.WriteLine(Math.Sqrt(n));
        }

        // int tipli sayının tamsayı karekökünü bulur
        public static int Isqrt(int num)
        {
            if (0 == num) { return 0; }  // Sıfıra bölmeyi önler
            int n = (num / 2) + 1;       // İlk tahmin
            int n1 = (n + (num / n)) / 2;
            while (n1 < n)
            {
                n = n1;
                n1 = (n + (num / n)) / 2;
            }
            return n;
        }
    }
```

Çıktı

```
35
35,1283361405006
```

Bazan hazır fonksiyonlardan yararlanarak kendi fonksiyonumuzu yaratabiliriz.

```
using System;

namespace Methods
{
```

```
    class LogaritmaBul
    {
        static int n = 1234;
        static void Main(string[] args)
        {
            Console.WriteLine(Math.Log(n));
        }

        static double LogBul(double y)
        {
            return Math.Log(y);
        }
    }
}
```

Çıktı

```
7,11801620446533
```

Uygulamalar

Matematik derslerinde verilen sayıların en küçük ortak katını (ekok) ve en büyük ortak bölenini (ebob) bulmak ilköğretimin önemli konularındandır. Şimdi bunu C# nasıl bulacağımızı gösterelim.

Ekok

```
/* a ve b ekok bulunacak sayılar */
  int ekokBul(int a,int b)
  {
    int n;
    for(n=1;;n++)
    {
        if(n%a == 0 && n%b == 0)
          return n;
    }
  }
```

```
using System;

namespace Metotlar
{
    public class MathMetotları
    {
        static int n, m, ekok;
        static string s;
        static void Main(string[] args)
        {
            Console.WriteLine("Lütfen ekok 'u bulunacak sayıların ilkini
giriniz:");
            s = Console.ReadLine();
            n = Int32.Parse(s);
```

```
            Console.WriteLine("Lütfen ekok 'u bulunacak sayıların
ikincisini giriniz:");
            s = Console.ReadLine();
            m = Int32.Parse(s);
            Console.WriteLine(ekokBul(n, m));
        }

        /* a ve b ekok'u bulunacak sayılar */
        static int ekokBul(int a, int b)
        {
            int n;
            for (n = 1; ; n++)
            {
                if (n % a == 0 && n % b == 0)
                    return n;
            }
        }
    }
}
```

Çıktı

```
Lütfen ekok 'u bulunacak sayıların ilkini giriniz:
35
Lütfen ekok 'u bulunacak sayıların ikincisini giriniz:
15
105
```

Aşağıdaki program `ekok` bulmak için *recursive* metot kullanmaktadır.

EkokBul02.cs

```
using System;

namespace Metotlar
{
    public class EkokBul02
    {
        static int x, y, ekok;
        static string s;
        static void Main(string[] args)
        {
            Console.WriteLine("Lütfen ekok 'u bulunacak sayıların ilkini
giriniz:");
            s = Console.ReadLine();
            x = Int32.Parse(s);
            Console.WriteLine("Lütfen ekok 'u bulunacak sayıların
ikincisini giriniz:");
            s = Console.ReadLine();
           y = Int32.Parse(s);
            Console.WriteLine(EkokBul(x, y));
        }
```

```
        static int EkokBul(int x, int y)
        {
            int prod;
            if (y % x == 0)
                return y;
            else
            {
                prod = x * y;
                while (x != y)
                {
                    if (x > y)
                        x = x - y;
                    else
                        y = y - x;
                }
                return EkokBul(y, prod / x);
            }
        }
    }
}
```

Çıktı

```
Lütfen ekok 'u bulunacak sayıların ilkini giriniz:
75
Lütfen ekok 'u bulunacak sayıların ikincisini giriniz:
45
225
```

Aşağıdaki program iki tamsayının en büyük ortak bölenini (EBOB) bulur.

```
using System;

public class EbobHesabı
{
    static int EbobBul(int a, int b)
    {
        int kalan;

        while (b != 0)
        {
            kalan = a % b;
            a = b;
            b = kalan;
        }

        return a;
    }

    static int Main(string[] args)
    {
        int x, y;

        Console.WriteLine("Ebob'i bulunacak sayıları giriniz : ");
```

```
        Console.Write("Sayı 1: ");
        x = int.Parse(Console.ReadLine());
        Console.Write("Sayı 2: ");
        y = int.Parse(Console.ReadLine());

        Console.WriteLine();
        Console.WriteLine("{0} ile {1} sayılarının ebob'ni: {2}", x, y,
EbobBul(x, y));

        return 0;
    }
}
```

Çıktı

```
Ebob'i bulunacak sayıları giriniz :
Sayı 1: 124
Sayı 2: 68

124 ile 68 sayılarının ebob'ni: 4
```

Aşağıdaki program ebob bulmak için *recursive* metot kullanmaktadır.

EbobHesabı02.cs

```
using System;

public class EbobHesabı
{
    static int EbobBul(int a, int b)
    {
        int c;
        while (a % b != 0)
        {
            c = a % b;
            a = b;
            b = c;
        }
        return b;
    }

    static int Main(string[] args)
    {
        int x, y;

        Console.WriteLine("Ebob'i bulunacak sayıları giriniz : ");
        Console.Write("Sayı 1: ");
        x = int.Parse(Console.ReadLine());
        Console.Write("Sayı 2: ");
        y = int.Parse(Console.ReadLine());

        Console.WriteLine();
        Console.WriteLine("{0} ile {1} sayılarının ebob'ni: {2}", x, y,
EbobBul(x, y));
```

```
            return 0;
        }
    }
```

Çıktı

```
Ebob'i bulunacak sayıları giriniz :
Sayı 1: 128
Sayı 2: 64


128 ile 64 sayılarının ebob'ni: 64
```

İkiden çok sayının `ebob` 'ni bulmak için, `EbobBul()` metodunu ve birleşme (associative) özeliğini kullanmak yetecektir. Örneğin, üç sayının `ebob` 'ni bulmak için,

```
EBOB(a, b, c) = EbobBul(EbobBul(a,b), c)
```

birleşme özeliğini kullanabiliriz:

EBOB.cs

```
using System;

public class EbobHesabı
{
    static int EbobBul(int a, int b)
    {
        int c;
        while (a % b != 0)
        {
            c = a % b;
            a = b;
            b = c;
        }
        return b;
    }

    static int Main(string[] args)
    {
        int x, y, z;

        Console.WriteLine("Ebob'i bulunacak üç sayıyı giriniz : ");
        Console.Write("Sayı 1: ");
        x = int.Parse(Console.ReadLine());

        Console.Write("Sayı 2: ");
        y = int.Parse(Console.ReadLine());

        Console.Write("Sayı 3: ");
        z = int.Parse(Console.ReadLine());
```

```
        Console.WriteLine();
        Console.WriteLine("{0}, {1} ve {2} sayıları için ebob : {3}", x,
y, z, EbobBul(EbobBul(x, y), z));

        return 0;
    }
}
```

Çıktı

```
Ebob'i bulunacak üç sayıyı giriniz :
Sayı 1: 24
Sayı 2: 48
Sayı 3: 64


24, 48 ve 64 sayıları için ebob : 8
```

Alıştırmalar

`string`'den tamsayıya dönüşüm:

```
using System;

namespace Metotlar
{
    class Dönüşümler
    {
        static void Main(string[] args)
        {
            string s = "1234";
            int  n = Int32.Parse(s); Console.WriteLine(n);
            int  m = int.Parse(s);   Console.WriteLine(m);
            int  p = short.Parse(s); Console.WriteLine(p);
            long q = long.Parse(s);  Console.WriteLine(q);
            Console.WriteLine();
        }
    }
}
```