

EĞİTİM :

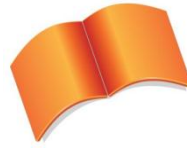
DİZİLER VE METOTLAR

Bölüm :

**Diziler ve Koleksiyonlara
Giriş**

Konu :

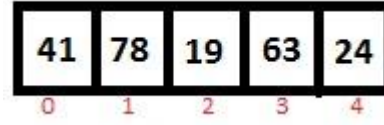
**Dizi Nedir? Neden İhtiyaç
Duyarız?**



Microsoft Türkiye

Açık Akademi

Diziler aynı tipte değerlerin bir arada taşınmasını sağlayan referans tipli programlama öğeleridir.



Diyelim ki 30 öğrencinin notlarını tutmak istiyorsun. Tip olarak **byte** işini görecektir (0 – 255 aralığında değer tutabilir). Bu programlama görevini yerine getirebilmek için byte tipinde 30 tane değişkene ihtiyaç olur. Ancak bu kullanım, takip etmesi oldukça güç durumlara yol açabilir. Bunun yerine byte tipinde 30 elemanlı bir dizi oluşturmak ve öğrencilerin notlarını bu dizide tutmak daha etkin bir çözümdür.

Ayrıca program içerisinde 30 kişinin notunu bir yerden başka bir yere taşımak da daha kolay olacaktır.

- Dizi tanımlanırken ona bir isim verilir. Ancak diziye atılan her bir eleman için ayrı ayrı değişken ismi verilmesine gerek yoktur.
- Diziler, elemanlarına ismiyle değil indeks sıralarıyla erişim sunar.
- Diziye atılan her bir eleman otomatik olarak 0'dan başlayıp 1'er 1'er artan bir indeksleme sistemi ile kaydedilir. Daha sonra dizi elemanlarına erişmek istediğinde indeksler ile erişebilirsiniz.
- Örneğin öğrencileri sınıf listesindeki sıralarına göre diziye atıp bu sıraya bakarak istediğin nota erişebilirsiniz.
- Diziler referans tiplidirler. Dolayısıyla diziye atılan veriler belleğin **heap** bölgesinde tutulurken, diziye erişim için oluşturulan değişken belleğin **stack** bölgesinde tutulur.

Not : Heap ve Stack, yazdığın .NET uygulamalarında kullanılan değişkenlerin ve tanımlanan nesnelerin geçici bellek (RAM) üzerinde saklandığı alanlardan ikisidir ve en temel olanlarıdır. Bu bölgeler .NET framework tarafından otomatik olarak yönetilirler. Geliştirici bu bölgelere doğrudan müdahale etmez. Kullanılan veri tipleri, önceden tanımlı kurallara göre belleğin ilgili alanında saklanırlar.

Stack, daha küçük bir alana sahiptir. Burada nispeten basit tipli verilerin saklanması amaçlanır. Değer tipli değişkenler, belleğin stack bölgesinde tutulur. (int, double, DateTime vb.)

Belleğin heap bölgesi, daha karmaşık verilerin saklanması için bulunmaktadır. Stack bölgesine göre bellekte daha büyük miktarda yer kaplar. Referans tipli değişkenler, belleğin heap bölgesinde tutulurlar.(string, object, Console vb.)

EĞİTİM :

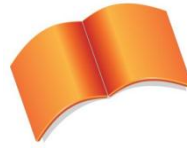
DİZİLER VE METOTLAR

Bölüm :

**Diziler ve Koleksiyonlara
Giriş**

Konu :

Dizileri Kullanmak



Microsoft Türkiye

Açık Akademi

Dizi Değişkenleri Tanımlamak

Dizi tanımlama söz dizimi, değişken tanımlama söz dizimi ile benzerdir. Bir dizi tanımlarken, dizi içerisinde hangi tipte değerler saklayacağını belirtmelisin. İkisini ayıran detay, dizide tipin sonuna koyulan köşeli parantezlerdir.

Örneğin **sayilarim** adında int tipinde bir dizi tanımlaması aşağıdaki gibi yapılır:

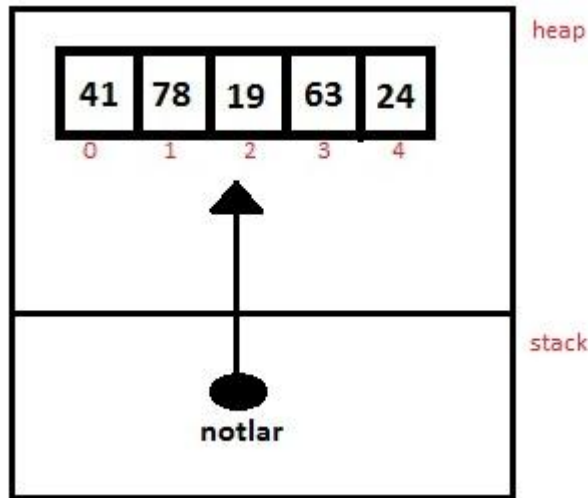
```
int[] sayilarim;
```

Burada tanımlanan tip, sadece önceden tanımlı tiplerden birisi (int, double vb.) olmak zorunda değildir. Temel sınıf kütüphanesindeki herhangi bir tipte olabileceği gibi kendi yazdığın tipte bir dizi de tanımlayabilirsin.

Dizi değişkenlerine çoğul adlar verilirse, bir diziye işaret ettiği daha rahat anlaşılabilir.

Dizi Örnekleri Oluşturmak

- İçerisinde tutulacak elemanların tipi ne olursa olsun diziler, referans tiplidirler. Yani dizi değişkeni belleğin **stack** bölgesinde, dizinin kendisi belleğin **heap** bölgesinde yer alır.



- Dizilerin belleğe çıkarılması için **new** anahtar kelimesi ile oluşturulmaları gerekir.
- Diziler oluşturulurken, kaç elemanlı oldukları bilinmek zorundadır. Bellekte, bu bilgiye göre yer açılır. Programın ilerleyen satırlarında bu alana belirtilen miktardan fazla eleman girilemez. Girilmeye çalışıldığında boyut dinamik olarak artmaz. Ortama hata fırlatılır.
- Yine dizilerin referans tipli olmasından ötürü, dizi üyeleri kendi tiplerinin varsayılan değerlerini alırlar. Daha sonra bu değerler, ihtiyaçlar doğrultusunda değiştirilir. Öyleyse 10 elemanlı bir bool dizisi oluşturulduğunda bu dizinin 10 elemanının değeri de otomatik olarak false (bool tipinin varsayılan değeri) şeklinde belirlenir.

Bu şekilde 4 elemanlı int tipli bir dizi oluşturabilirsin.

```
sayilarim = new int[4];
```

Dizinin eleman sayısını sabit bir değer olarak vermek zorunda değilsin. Tek kural var; o da dizi oluşturulduğunda kaç elemanlı olacağı bilinmelidir. Bu yüzden, dizi örneği farklı yollar kullanılarak oluşturulabilir. Örneğin dizinin eleman sayısı çalışma zamanında belirlenebilir:

...

```
Console.Write("Lütfen öğrenci sayınızı giriniz : ");  
int boyut = int.Parse(Console.ReadLine());  
sayilarim = new int[boyut];  
Console.WriteLine("{0} elemanlı diziniz oluşturuldu...",boyut);
```

...

Dizilere Başlangıç Değerlerini Vermek

- Bir dizi oluşturulduğunda dizinin içi, oluşturulduğu tipin varsayılan değerleriyle doldurulur. Dizinin bu davranışı değiştirilebilir ve başlangıçta istenilen değerlerle belleğe çıkarılabilir. Bunu gerçekleştirmek için, dizi oluşturulma söz diziminin sonunda küme parantezleri kullanılır. Dizide belirtilen tipte ve eleman sayısı kadar değeri, süslü parantezler içine, aralarında virgül olacak şekilde yazman gerekir. Örnek olarak **sayilarim** adlı dizi, 4,9,7 ve 3 değerleri ile birlikte oluşturulabilir.

```
sayilarim = new int[4] { 4,9,7,3};
```

- Burada dizi boyutunun 4 elemanlı olacağı bildirildiği için süslü parantezler içerisine 4 elemandan az ya da fazla değer girilmesi durumunda derleme zamanı hatası alınır. Birkaç veri tipi için bu söz diziminin örnek kullanılışı şöyledir:

```
string[] sehirler = new string[] { "Antalya","İstanbul","Ankara" };  
double[] oranlar = new double[3] { 0.32 , 0.45 , 0.56 };  
char[] alfabe = new char[3] { 'a','b','c'};
```

- Dizi oluşturulurken eğer elemanların ilk değerlerini veriyorsan, eleman sayısı belirtmek zorunda değilsin. Derleyici, başlangıç değerlerindeki eleman adedine bakarak dizinin eleman sayısını hesaplar.

```
string[] sehirler = new string[] { "Antalya","İstanbul","Ankara" };
```

- Dizi oluşturulurken eğer elemanlarını veriyorsan, **new** anahtar kelimesi ve veri tipi kullanımı ihmal edilebilir. Derleyici, başlangıç değerlerinin adedine bakarak dizinin eleman sayısını otomatik olarak hesaplar.

```
string[] sehirler = { "Antalya","İstanbul","Ankara" };
```

Her Bir Dizi Elemanına Erişmek

- İstenilen dizi elemanına erişmek için dizi üzerinden ilgili elemanın indeks sırası verilir. Örneğin **sayilarim** adıyla oluşturulup değerleri verilen dizinin son elemanına erişmeye çalıştığını düşünelim. Burada dikkat edilmesi gereken nokta; **C#'da dizi mantığındaki her şeyin indeks olarak 0 değeri ile başlamasıdır.** Tıpkı ekrana bir şey yazdırırken kullanılan yer tutucularda (placeholder) olduğu gibi ({0}). O yüzden aşağıdaki dizinin elemanları ve indeks sıraları şöyledir :
 - 0. indeksli elemanı 4
 - 1. indeksli elemanı 9
 - 2. indeksli elemanı 7
 - 3. indeksli elemanı ise 3

```
int[] sayilarim = new int[4] { 4,9,7,3};  
int sayi2 = sayilarim[1];  
Console.WriteLine("Dizinin {0}. indeksli elemanı : {1}",1,sayi2);
```

- Dizi elemanlarının değerleri tanımlandıktan sonra değiştirilebilir. Dizinin ikinci indeksli elemanı olan 7 değerini aşağıdaki kod parçasında görüldüğü gibi 10 olarak değiştirebilirsin.

```
sayilarim[2] = 10;  
// ya da  
int baskaSayi = 10;  
sayilarim[2] = baskaSayi;
```

- Bir diziye değerleri verilmeden sadece eleman sayısı söylenerek (varsayılan değerleri ile) oluşturulduktan sonra dizi elemanlarını tek tek de verebilirsin.

```
int[] sayilarim = new int[4];  
sayilarim[0] = 3;  
sayilarim[1] = 5;  
sayilarim[2] = 7;  
sayilarim[3] = 9;
```

Dizinin elemanlarına erişimde 0'dan başlayan indeksler kullanıldığı için dizinin son indeksi, **eleman sayısı -1** olarak ele alınır. Eğer yanlışlıkla üç elemanlı bir dizinin üçüncü indeksli elemanına erişmeye çalışırsan **IndexOutOfRangeException** hatası ile karşılaşırısın.

```
int[] sayilarim = new int[4] { 4, 9, 7, 3 };
Console.WriteLine(sayilarim[4]); //HATA: Bu diziye erişim için kullanılacak
index 0 ile 3 arasında olmalıdır.
```

Bir Dizinin Eleman Sayısını Elde Etmek

Bir dizinin eleman sayısını, dizi değişkeni üzerinden çağıracağın **Length** özelliği ile elde edebilirsin.

```
Console.WriteLine("sayilarim dizisinin eleman sayısı : {0}",
sayilarim.Length);
```

Bir Dizi İçerisindeki Bütün Elemanları Elde Etmek

Bir dizideki eleman sayısı dizinin **Length** özelliği ile elde edilebildiğine göre, **for** döngüsü kullanılarak dizinin 0'nci indeksi ile **eleman sayısı -1**'nci indeksi arasındaki bütün değerler elde edilebilir.

```
int[] sayilarim = new int[4] { 3,5,7,9 };
for (int i = 0; i < sayilarim.Length; i++)
{
    Console.WriteLine("{0}.eleman = {1}",i + 1,sayilarim[i]);
}
```

Yukarıdaki döngüde ekrana her bir elemanın dizideki sırası (indeks + 1) ve değeri yazdırılmaktadır.

- Döngü değişkeni önce sıfır değerini alır.
- Ekrana mevcut elemanın sıra numarası olarak 1'i yazdırır (0 + 1).
- Yanına dizideki sıfırıncı indeksli elemanın değeri olan 3'ü yazdırır.

Burada dikkat edilmesi gereken noktalardan biri; döngü değişkeninin başlangıç değeri olarak 1'le değil 0'la başlamasıdır.

Diğer nokta ise döngünün mantıksal koşul deyiminin, dizinin boyutu olan 4 için çalıştırılmamasıdır. Bu durum, koşul deyiminin $i \leq \text{sayilarim.Length}$ yerine $i < \text{sayilarim.Length}$ olarak yazılmasından kaynaklanır.

- Döngü bu şekilde dört değer için de çalışır ve dizideki bütün elemanların değerleri elde edilmiş olur.

Not : Ayrıca indeks değişkenini kendin tanımlayarak **while** döngüsü yardımıyla, dizi içerisindeki bütün elemanlara ulaşman mümkündür.

Dizi Elemanlarını foreach Döngüsü ile Elde Etmek


C#, dizilerin içerisinde dönüp değerlerini elde etmek için bir yol daha önerir: **foreach** döngüsü.

```
foreach (int gecici in sayilarim)
{
    Console.WriteLine(gecici);
}
```

- **foreach** döngüsünde tanımlanan döngü değişkeni otomatik olarak dizideki her elemanı sırayla elde eder.
- Döngünün ilk iterasyonunda (ilk turunda) belleğe çıkarılan değişkenin tipi, dizinin elemanları ile aynı tipinde olmalıdır.
- **foreach**, arka tarafta kendi algoritması ile her iterasyonda dizinin bir elemanını okuyup bellekteki geçici değişkene atar. Ardından döngü bloğu içerisinde değişkenin o anki değeri okunur.
- **foreach**, dizideki değerleri okuma işini **0.** indeksten başlayıp **eleman sayısı - 1**'nci indekse kadar sırayla yapar. **for** döngüsünde düşünülmesi gereken ayrıntılar **foreach**'de olmadığı için dizinin bütün elemanlarını okumada tercih edilebilir.

Yine de bir dizinin elemanları elde edilmek istendiğinde **for** döngüsünün daha kullanışlı olduğu alanlar vardır.

- **foreach** deyimi her zaman bütün diziyi döner. Eğer sadece dizinin belli bir bölümü elde edilmek isteniyorsa (mesela yarısı) ya da belli değerler atlanmak isteniyorsa (Mesela her 3 elemandan biri) bunu **for** kullanarak yapmak çok daha kolaydır.
- **foreach** deyimi her zaman dizinin **0.** indeksinden **eleman sayısı - 1**'nci indeksine doğru çalışır. Dolayısıyla eğer dizinin değerleri tersten okunmak istenirse **for** döngüsünü kullanmak gerekir.
- Eğer döngü bloğu içerisinde sadece dizideki elemanın değeri değil indeks sayısı da istenirse, bu desteği sağlamak için **for** döngüsü kullanılmak durumundadır. **foreach** döngüsü ile bu desteği sağlamak için ekstra bir indeks değişkeni oluşturulup her turda arttırım işlemleri yapılabilir.
- Eğer dizi elemanlarının değerleri bir döngü içerisinde güncellenmek istenirse, **for** döngüsü kullanılmak zorundadır. Çünkü **foreach** deyimindeki döngü değişkeni, dizideki her bir elemanın sadece okunabilir kopyasını elde eder. Bu yüzden de dizi elemanlarının değeri **foreach** döngüsü içerisinde değiştirilemez.

 Foreach döngüsü, sadece ileri yönlü ve yalnız okunabilir bir öteleme hareketine izin verdiğinden, çoğu durumda **for** döngüsüne göre daha performanslı çalışır.

Örnek Uygulama

Aşağıda örnek bir uygulama bulunmaktadır. Öncelikle kullanıcıdan kaç öğrenci için not gireceği bilgisi alınır. Alınan bu bilgi ile bir dizi oluşturulur. Dizinin dinamik oluşan eleman sayısı kadar kullanıcıdan değer alınır. Kullanıcıdan alınan değerler, dizinin elemanları olarak sırayla kaydedilir. Ardından dizinin boyutu ve elemanları toplu olarak kullanıcıya gösterilir:

```
...

//Kullanıcıdan oluşturulacak dizinin eleman sayısı alınır...
Console.Write("Lütfen öğrenci sayınızı giriniz : ");
```



```

byte elemanSayisi = byte.Parse(Console.ReadLine());
byte[] notlar = new byte[elemanSayisi];
//Kullanıcıdan alınıp döngü içerisinde diziye girilen değerleri tutacak //olan
değişken tanımlanır.
byte alinanEleman;

//Döngü içerisinde kullanıcıdan sırayla notlar alınıp bir değişken //aracılığıyla diziye
verilir.

for (byte i = 0; i < notlar.Length; i++)
{
    Console.Write("Lütfen {0}. öğrencinin notunu girip ENTER'a basınız :
",i + 1);
        alinanEleman = byte.Parse(Console.ReadLine());
        notlar[i] = alinanEleman;
    }
    Console.WriteLine("Veri girişi tamamlandı. Devam etmek için
ENTER'a basınız...");
    Console.ReadLine();
    //Kullanıcıdan girişleri aldıktan sonra verileri görmek isteyip //istemediği sorulur.
    etiket:
    Console.Write("Girilen notları görmek ister misiniz? [E] [H] : ");
    string devamMi = Console.ReadLine().ToLower();
    //Kullanıcı eğer [E] yazarsa, ona öğrenci sayısını ve notlarını gösterilir.
    //Kullanıcı eğer [H] yazarsa, program sonlandırılır.
    //Kullanıcı eğer [E] ve [H] dışında bir değer girerse, tercihi tekrar //sorulur...
    switch (devamMi)
    {
        case "e":
            Console.WriteLine("\n{0} öğrencinin notları sırasıyla
 aşağıdadır:\n",notlar.Length);
            foreach (byte gecici in notlar)
            {
                Console.Write(gecici + " ");
            }
            break;

        case "h":
            break;

        default:
            goto etiket;
    }

```

EĞİTİM :

DİZİLER VE METOTLAR

Bölüm :

**Diziler ve Koleksiyonlara
Giriş**

Konu :

Koleksiyonlara Giriş



Microsoft Türkiye

Açık Akademi

Diziler çok yararlı programlama nesneleridir. Fakat bazı sınırlamaları vardır. Bunlardan birincisi dizilerin sadece tek tipten değerler kabul edebilmesidir. Ancak bu sorun yine dizilerle çözümlenebilir. Biliyorsun ki **object** tipinden bir değişken .NET dünyasındaki bütün tipleri üzerinde taşıyabilmektedir. Öyleyse object tipinden oluşturulan bir dizi ile tip sınırlaması sorunu aşılabılır ve tek bir dizi içerisinde istenilen tipte değerler taşınabilir:

```
object[] karmaDizi = new object[] { 'A',1,true,"Emrah",0.45 };
```

Dizilerle ilgili ikinci sıkıntı ise dizilerin boyutudur. Bir dizinin boyutu, tanımlama sırasında verildikten sonra **Array** sınıfının **Resize** adlı metodu ile değiştirilebilir. Ancak diziye her eleman eklenmesinde ya da diziden her eleman çıkarılışında bu üyenin çağırılması gerekmektedir. Bu da sıkıntıcı verici bir durumdur.

C#, birden fazla değeri topluca saklamak için bir programlama ögesi daha sunar: **Koleksiyonlar (Collections)**. **System.Collections** isim alanı altında yer alan birçok koleksiyon sınıfı vardır. Koleksiyon sınıfları, her halükarda object tipinden değer alan tiplerdir. Dolayısıyla her tipte değer kabul ederler. Ayrıca koleksiyonlar herhangi bir boyut sınırlaması olmayan diziler gibidir.

Bir koleksiyon sınıfı oluşturulurken referans tipli olduğu için **new** anahtar kelimesi kullanılır. Herhangi bir eleman sayısı bildirimi yapılmaz. Oluşturulduktan ve elemanları verildikten sonra istenildiği zaman herhangi bir eleman koleksiyondan çıkarılabilir veya yeni bir eleman eklenebilir. Koleksiyon içinde yapılan değişikliklerde boyut dinamik olarak değişir. Ayrıca bütün bunlar tip sınırlaması olmadan yapılır.

Yazacağın programlarda koleksiyonların kullanım alanları için şöyle gerçek hayat örnekleri verilebilir :

- *Web sitesi üzerinden özgeçmiş formu doldurma.* Bu senaryoda istenen sayıda e-posta adresi girişi yapılması ve önceden sayısı bilinmeyen e-posta adreslerinin tutulması.
- *Öğrenci bilgi sistemi uygulamasında ders seçimi.* Bir üniversite öğrencisi, web sitesi üzerinden istediği dersleri seçip çıkartırken, kod tarafında dersler bir koleksiyonda tutulabilir.
- *E-Ticaret uygulamasında alışveriş sepeti.* Ziyaretçi, sitede farklı sayfalarda gezerken (sayısı önceden bilinmesi mümkün olmayan) istediği ürünleri ekleyip çıkarabilmesi için bilgiler koleksiyon sınıflarında saklanabilir. (Not : Bu senaryoda sayfalar arası bilgilerin taşınmasını sağlamak için ek çözümlere de ihtiyaç olacaktır)

En sık kullanılan koleksiyon tiplerinden biri **ArrayList**'tir. Ayrıca SortedList, Stack ve Queue diğer koleksiyon sınıflarından bazılarıdır. Bir koleksiyonun oluşturulduğu, değer atanması, değerlerine erişilmesi ve bütün değerlerinin elde edilmesi aşağıdaki kod parçasında olduğu gibi gerçekleştirilebilir:

```
using System;  
using System.Collections;  
  
namespace Koleksiyonlar  
{  
    class ArrayListKullanimi
```

```

{
    public static void Main(string[] args)
    {
        //Koleksiyon sınıfı oluşturulur..
        ArrayList koleksiyonum = new ArrayList();
        //Koleksiyona object tipinden elemanlar eklenir.
        koleksiyonum.Add(1);
        koleksiyonum.Add("Emrah");
        koleksiyonum.Add(25);
        koleksiyonum.Add(new DateTime(1982,2,2));
        //Koleksiyon elemanlarına, dönüşüm (cast) yaparak erişilir. Çünkü
        koleksiyon içerisinde object tipi ile tutulurlar.
        string adi = (string) koleksiyonum[1];
        Console.WriteLine("*****Koleksiyondaki 1.indeksli elemana
erişildi*****\n" + adi);
        //Koleksiyondaki elemanların tümüne erişilir : herhangi bir döngü türü ile
        olabilir.
        Console.WriteLine("*****Koleksiyondaki tüm elemanlara
erişildi*****");
        foreach (object gecici in koleksiyonum)
        {
            Console.WriteLine(gecici);
        }

        Console.ReadLine();
    }
}

```

EĞİTİM :

DİZİLER VE METOTLAR

Bölüm :

**Diziler ve Koleksiyonlara
Giriş**

Konu :

Metot Nedir?



Microsoft Türkiye

Açık Akademi

Metot Nedir? Neden İhtiyaç Duyarız?

Metotlar, bir kod bloğuna isim verip, uygulamanın başka yerlerinde, o isimle çağırıp yeniden kullanılmasına izin veren programlama öğeleridir. Her metodun bir ismi ve bir kod bloğu vardır. Metodun adı, amacına uygun olarak verilmelidir. Örneğin, bir sınıftaki öğrencilerin başarı oranının hesaplamasını yapan bir metoda **BasariOraniHesapla** isminin verilmesi gibi. Verilen adla çağırıldığında çalışacak kodlar küme parantezi { } bloğu arasında yazılır. Bazı metotlara kod bloğu içerisinde kullanması için girdi olarak veri aktarılabilir. Ayrıca metot, içeride çalışan kodların sonucunda geriye bir bilgi döndürebilir. Çoğunlukla bir metot geriye bilgi döndürdüğünde, bu aynı zamanda metodun çalışmasının sonu olur.

- Metotlar çok temel ve güçlü nesnelerdir. Kodun yeniden kullanılabilirliğini sağlarlar. Ayrıca tek bir yerden değişiklik yaparak, kullanıldığı her yerde akış mantığını değiştirebilirler.
- Metotlar, sadece bir sınıf veya bir yapı içerisinde oluşturulabilirler. Bağımsız olarak bir isim alanı altına ya da sınıf veya yapı dışındaki bir tipin içerisine metot yazılamaz.

EĞİTİM :

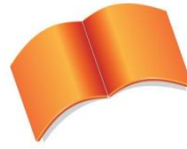
DİZİLER VE METOTLAR

Bölüm :

**Diziler ve Koleksiyonlara
Giriş**

Konu :

Metot Oluşturma Söz Dizimi



Microsoft Türkiye

Açık Akademi

Bir C# metodu yazarken takip edilmesi gereken söz dizimi şu şekildedir:

erisimBelirleyici niteleyici donusTipi MetotAdi (parametre listesi) (varsa) (varsa)
<pre>{ //Metot çağırıldığında işletilecek kodlar buraya yazılır. }</pre>

Bir C# metodu çağrılırken kullanılacak söz dizimi ise şudur.

MetotAdi(varsa parametre değerleri);

erisimBelirleyicisi, metotların dışında bir konudur. Bir metodun sınıf veya yapı üyesi olmasının sonucu belirtilmesi gereken bir anahtar kelimedir. C# dilinde beş erişim belirleyicisi vardır ve bu erişim belirleyiciler birer anahtar kelime ile temsil edilir. Burada erişim belirleyicilerin sadece iki tanesinden bahsedilecektir: Private ve Public.

Private erişim belirleyicisi metodun sadece içerisinde yer aldığı sınıf veya yapı üyeleri tarafından çağırılıp kullanılabilmesini sağlar. **public** erişim belirleyicisi ise metodun her yerden (kendi yazdığımız başka yapı ve sınıflar da dahil) çağırılıp mesini sağlar. Metot yazılırken erişim belirleyicisi belirtilmezse metodun erişim belirleyicisinin private olduğu varsayılır.

donusTipi, bir .NET tip adıdır. Metodun çalışması sonucu, kendisini çağırıp kullanana göndereceği verinin tipini belirler. Bu tip int, string gibi önceden tanımlı tiplerden birisi olabileceği gibi kendi yazdığımız her hangi bir tip de olabilir. Eğer geriye değer döndürmeyen bir metot yazılıyorsa bu kısma **void** kelimesi yazılmalıdır. Mesela bir metot, kod bloğu içerisinde bazı işler yaptıktan sonra sadece ekrana bir şeyler yazdırıyorsa, değer döndürmüyor demektir ve dönüş tipi **void** olarak tanımlanır. Başka bir metot, içeride yaptığı bir hesaplama sonucunu çağırıldığı ortama aktarıyorsa aktardığı bilginin veri tipi neyse, dönüş tipi de o olacaktır (Metot içerisinde iki int tipli değişkenin toplamı, metodu kullanan ortama **int** dönüş tipiyle bir değer gönderebilir.)

niteleyici, ileri seviye programlamada kullanılan **static**, **abstract**, **sealed**, **virtual** kelimelerinden birinin yerini tutar. (Bu anahtar kelimelerin kullanıldığı konular, eğitimin kapsamı dışında kaldığı için **static** hariç bu anahtar kelimeler anlatılmayacaktır.)

MetotAdi, metot çağrılırken kullanılacak olan isimdir. Bu isim verilirken takip edilecek kural ve öneriler değişkenlerinki ile benzerdir. Sadece öneriler kısmında adın baş harfinin büyük olması, kod yazım standartlarına destek vermek adına tavsiye edilir (Örneğin değişkenler için girisSayisi iken, metotlar için GirisSayisiniHesapla() olur).

parametre listesi, opsiyoneldir; yani kullanılabilir de kullanılmayabilir de. Ancak her iki durum için de metot adından sonra açılış ve kapanış parantezleri bulunmak zorundadır. Parametreler, metodun çalışması sırasında kullanılabilecek değerlerdir. Metodun yazım aşamasında sanki varmış gibi kabul edilerek kullanılırlar. Gerçek değerleri, metodun çağırıldığı yerde verilir. Bu sayede son derece esnek bir uygulama

geliştirme modeli sunarlar. Kullanılacak parametreler, metodun yazım aşamasında metot adından sonra açılan parantezin ardından sırayla tip adı ve değişken adı olacak şekilde tanımlanır. Parametreleri ayırmak için, iki parametre arasında virgöl kullanılır.

Metot bloğu, metot çağırıldığında çalışacak olan kodların yazılacağı yerdir ve süslü parantezlerle sınırlanır { }.

EĞİTİM :

DİZİLER VE METOTLAR

Bölüm :

**Diziler ve Koleksiyonlara
Giriş**

Konu :

**Metot Nasıl Yazılır ve Nasıl
Kullanılır?**



Microsoft Türkiye

Açık Akademi

Aşağıda basit bir metot örneği yer almaktadır:

```
using System;

namespace Metotlar
{
    class MetotKullanimi
    {
        public static void MesajYazdir()
        {
            Console.WriteLine("Merhaba C# metotları...");
        }

        public static void Main(string[] args)
        {
            MesajYazdir();

            Console.ReadLine();
        }
    }
}
```

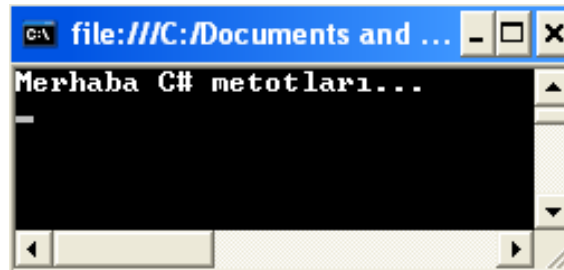
Basit bir metot örneği

Yukarıda, geriye değer döndürmeyen ve çağrıldığında sadece ekrana bir şeyler yazdıran **MesajYazdir** adında bir metot tanımlanmaktadır. Geriye değer döndürmediği için dönüş tipi olarak void anahtar kelimesi belirtilmiştir.

Bir metot sadece sınıf ya da yapı içerisine yazılabilir. Bu metoda baktığımızda **MetotKullanimi** sınıfının içerisinde yazıldığı ve dolayısıyla bu sınıfın bir üyesi olduğu görülür.

Ayrıca metodun, **static** niteleyicisi ile yazıldığı görülmektedir. Main metodunun bulunduğu sınıf içerisine bir metot yazılırsa, doğrudan metot adıyla erişim için static niteleyicisinin kullanılması gerekmektedir. Bu kuralı test etmek için, static kelimesini kaldırıp, uygulamayı yeniden çalıştırmayı deneyebilirsiniz. Bu durumda derleme zamanı hatası alınır. Bu kelimenin sürece kattığı gerçek anlam, sınıf kavramı detaylarıyla öğrenildiğinde daha kolay anlaşılacaktır.

MesajYazdir isimli metodun, Main() içerisinde çağırılması sonucu oluşan ekran çıktısı aşağıdadır. (Şu ana kadarki bütün örnekler Main() metodu içerisinde yazılmıştır. Bunun sebebi Main() 'in uygulamanın başlangıç noktası olmasıdır. Bu yüzden metodun çağırıldığı yer yine Main() 'in içidir).



Metot örneğinin çalışması

Metotların içerisinde yazılacak kodun satır sınırı yoktur. İstenilen programlama görevi, bir metot içerisinde kodlanabilir. Ancak satır sayısı çok fazla ise yapılan işleri farklı metotlara bölmek daha faydalı olacaktır. Yazılan herhangi bir kod bloğunu, uygulamanın birçok yerinde kullanma ihtiyacı olduğunu düşünelim. Bu kodları her ihtiyaç

duyulduğunda tekrar tekrar yazmak yerine, bir defa metod içerisine yazmak daha uygun olacaktır. Daha sonra ihtiyaç olan yerlerde, metodun adıyla çağrı yaparak bu kod bloğunun çalışması sağlanır. Bu durum, aynı zamanda program kodunun daha rahat okunabilmesine, optimize edilebilmesine ve bakımının kolayca yapılabilmesine de olanak sağlar. Örneğin uygulamanın çeşitli yerlerinde güncel tarih ve saat kullanıcıya gösterilmek istenirse ilgili kodu aşağıdaki gibi her yere yazmak tercih edilmez.

```
static void Main(string[] args)
{
    Console.WriteLine(DateTime.Now);

    //...

    Console.WriteLine(DateTime.Now);

    //...

    Console.WriteLine(DateTime.Now);

    Console.Read();
}
```

Bunun yerine güncel tarih ve saati gösteren kod bir metod içerisine yazılır ve istenilen yerde metod çağrılır. (Kodun Tekrar Kullanılabilirliği – Code Reusability)

```
...

static void TarihGoster()
{
    Console.WriteLine(DateTime.Now);
}

public static void Main(string[] args)
{
    TarihGoster();

    //...


    TarihGoster();

    //...

    TarihGoster();

    Console.Read();
}

...
```

 Metod içerisindeki kod biraz daha uzun olduğunda tabii ki daha anlamlı olacaktır.

Bu yaklaşımın bir kazancı daha vardır. Kodda bir değişiklik yapılmak istendiğinde eğer metod kullanılmamış ise; bu değişikliğin kodun yazıldığı her yerde ayrı ayrı yapılması gerekir. Fakat metod kullanılmışsa; değişiklik sadece bir yerde yapılır ve metodun kullanıldığı her nokta bu değişiklikten otomatik olarak etkilenir. (Kodun bakım kolaylığı – Code Maintainability)

Buraya kadar kendi metotlarımızı nasıl yazıp kullanabileceğimizi inceledik. Aynı zamanda eğitimin bu bölümüne gelene kadar faydalanılan metotlar var. Örneğin programın başlangıç noktası olan Main() metodunu incelediğimizde, normal bir metotta olması gereken herşeye sahip olduğu görülür. Başka bir örnek olarak metotların bir sınıf(class) ya da yapı(struct) üyesi olabileceği bilgisinden yola çıkarak; WriteLine() ve ReadLine() metotları göz önüne alınabilir. Console sınıfının üyesi olan bu metotlar ve önceden tanımlı sınıf ve yapılar içerisinde yer alan pek çok diğer metot, .NET platformu için geliştirilmiştir. Bu metotlar, programcıların ihtiyacı olan fonksiyonellikleri karşılamak üzere tasarlanmışlardır.

EĞİTİM :

DİZİLER VE METOTLAR

Bölüm :

**Diziler ve Koleksiyonlara
Giriş**

Konu :

Return Anahtar Kelimesi



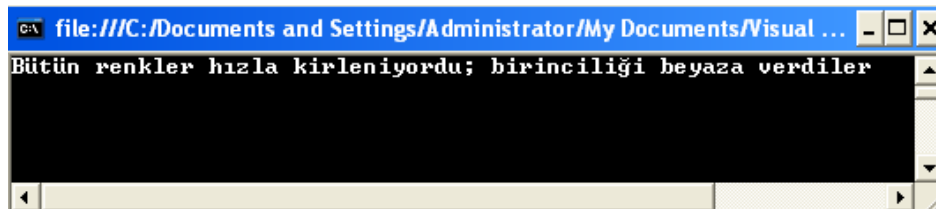
Microsoft Türkiye

Açık Akademi

Bir metod içerisindeki kodların çalışması varsayılan olarak, blok sonuna yani küme parantezi kapanışına gelindiğinde sonlanır. Ardından kod akışı metodun çağırıldığı yere geri döner. Eğer metodun son deyim beklenmeden sonlanması istenirse **return** anahtar kelimesi method içerisinde herhangi bir noktada kullanılabilir.

```
...  
  
static void GununSozunuVer()  
{  
    Console.WriteLine("Bütün renkler hızla kirleniyordu; birinciliği beyaza verdiler");  
    return;  
    Console.WriteLine("Sözün sahibi : Özdemir Asaf");  
}  
  
public static void Main(string[] args)  
{  
    GununSozunuVer();  
  
    Console.ReadLine(); //Uygulama F5 ile çalıştırıldığında konsol  
                        //ekranının hemen kapanmaması için  
                        yazılır.  
}  
...
```

GununSozunuVer metodu, Main() metodu içerisinde çağırılıp çalıştırılır. Elde edilen çıktıda; return anahtar kelimesinden sonra ekrana yazdırılmak istenen **"Sözün sahibi : Özdemir Asaf"** kısmı yer almaz. Çünkü return kelimesinin görülmesiyle birlikte metodun çalışması sonlanır.



return anahtar kelimesinin kullanımı

Bu örnekteki gibi bir kullanımın anlamının olmadığı düşünülebilir. Çalışmayacağını bile bile neden return anahtar kelimesinden sonra kod yazalım ki diye düşünebilirsiniz. O yüzden return anahtar kelimesini, if ya da switch gibi koşullu deyim bir parçası olarak kullanmak daha uygun olacaktır. Yani belli bir koşul sağlanırsa metodun sonlanması beklenmeden çıkılmak istendiğinde **return** anahtar kelimesinden faydalanılabilir.

```
static void OduVer()  
{  
    Console.WriteLine("Eğer ortalaman 70 in üzerinde ise Antalya  
    tatilini hakkettin.\n");  
    int ortalama = 75;  
    if (ortalama > 70)  
    {  
        Console.WriteLine("Ortalaman : {0}...\nNe zaman istersen  
        tatile çıkabilirsin.\nÇünkü çok çalışarak bunu hakkettin...",  
        ortalama);  
    }  
}
```

```

        return;
    }

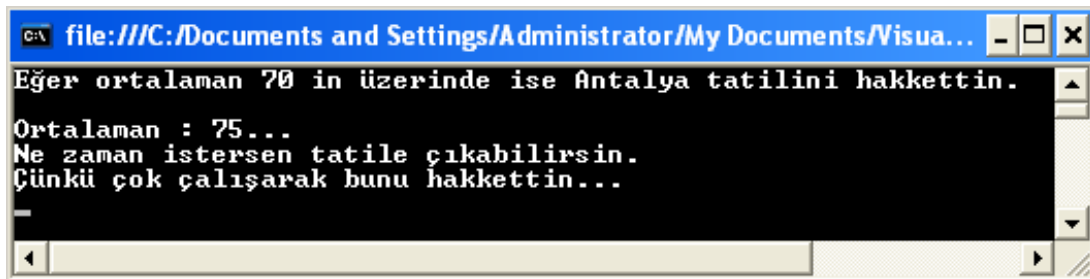
    Console.WriteLine("Ortalaman : {0}...\nMaalesef tatil fırsatını
    kaçırdın.\nÇok çalışmak lazım, çok...",ortalama);
}

public static void Main(string[] args)
{
    OdulVer();

    Console.ReadLine();
}

```

Yukarıdaki kod bloğunda ortalama değişkeninin alacağı değer 70'in üzerinde ise ekrana bir şeyler yazılacak. Ardından **return** deyimi çalışacağı için metodun çalışması sonlanacaktır. **return** deyiminden sonra yazılan kodlar çalışmaz. Eğer ortalama değişkeninin değeri 70 ya da 70'den küçük ise kod akışı, **return** deyiminin bulunduğu if bloğuna girmez. Bundan dolayı **return**'den sonraki kodlar çalışır. Bu kod, ortalama değişkeninin değeri ile oynanarak test edilebilir.



return anahtar kelimesi ile metodu, küme parantezi kapanışına gelmeden sonlandırmak

return ile Bir Değer Döndürmek

Şu ana kadar öğrendiklerin doğrultusunda, **return** deyimini metodun sonunda kullanmanın çok da anlamlı olmayacağını düşünebilirsin. Zaten metodun sonuna gelindiğinde çalışması sonlanır. Fakat, metodun çalıştığında kendisini çağıran kullanıcıya bir değer göndermesi gerektiğinde, **return** deyiminin metodun sonunda kullanılması daha anlamlı olur. Metodun çağrıldığı yerde kullanılmak üzere, metod içerisinde yapılan işlerin sonucunda oluşan bir değere ihtiyaç duyulabilir. Bu değeri ele alıp başka işlemlerin yapılması istenebilir. İşte bu değer metodun sonunda **return** ifadesinden hemen sonra kullanılarak, çağrı yapılan yere gönderilebilir. Örneğin, metod içerisinde tanımlanan ve değeri verilen iki değişkenin toplamının hesapladığını düşünelim. Yapılan işlemlerin arkasından **return** deyimi ile sonuc değişkeninin değeri, metodun kullanıldığı yere döndürülebilir. Burada dikkat edilmesi gereken bir nokta vardır. Hatırlarsanız bu konunun başında **metot söz dizimi** anlatılırken, **donustipi** kavramından bahsedilmiştir ve şu ana kadar yazılan metotlarda sadece konsol ekranına birşeyler yazdırıldığı için sadece **void** anahtar kelimesi kullanılmıştır. Az önce bahsedilen senaryo için ise dönüş tipi **void** değil; **return** ile döndürülecek değere uygun bir veri tipi olmalıdır:

```

static int Hesapla()
{
    int a = 2;
    int b = 21;
    int sonuc = a + b;

    return sonuc;
}

```

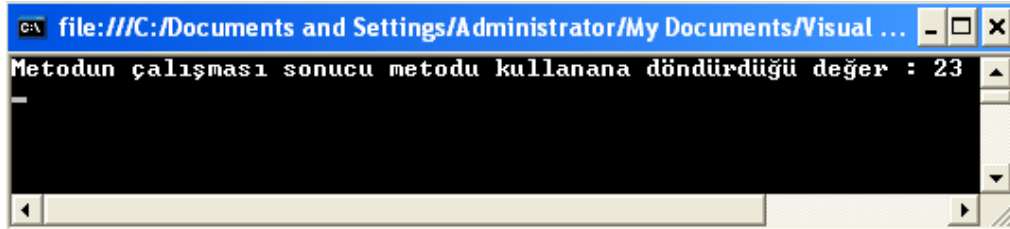
Metot, yukarıda belirtildiği gibi iki değişkenin değerini toplayıp üçüncü bir değişkene atar ve kendisini çağıranın bu değeri elde edebilmesi için **sonuc** isimli değişkeni döndürür. Metot dışarıya bir değer döndürdüğü ve **sonuc** isimli değişkenin tipi int olduğu için metodun dönüş değeri artık void olamaz. Dönüş değeri int olarak tanımlanmalıdır.

Bir metodun dönüş tipini belirlerken, değişkenlerde yer alan büyük tür ve küçük tür arasındaki dönüşüm kuralları geçerlidir.

Şimdi de bu metodun Main() içerisinde çağrılıp, dönüş değerinin ele alınışını görelim:

```
public static void Main(string[] args)
{
    int metottanGelenDeger = Hesapla();
    Console.WriteLine("Metodun çalışması sonucu metodu kullanana
döndürdüğü değer : {0}",metottanGelenDeger);

    Console.ReadLine();
}
```



return anahtar kelimesi ile dışarıya değer döndürmek

Dönüş değeri **void** iken, ihtiyaç duyulduğu yerde sadece metodu çağırmak yeterlidir.

```
public static void Main(string[] args)
{
    Hesapla();

    Console.ReadLine();
}
```

Fakat, dönüş değeri int olan Hesapla() metodunda, bu şekildeki kullanımın bir anlamı yoktur. Çünkü metodu yazarken amacımız, metodun çağrıldığı yere, yapılan işlemin sonucunu döndürmektir. Sadece metodu çağırmakla, içerde yapılan işlemin sonucunu elde edemeyiz. Bu yüzden, metodun geri dönüş değeri ile aynı veri tipinde (burada int) bir değişken oluşturmak ve değişkenin başlangıç değerini metottan almak gerekir. Yukardaki kodda bu değişken, **metottanGelenDeger** dir. Değişken ilk değer ataması sırasında, atama operatörünün sağında bir metot bulunması ilginç gelebilir. Geri dönüş değeri olan bir metodun kullanıcı için anlamı, döndürdüğü değerdir ve bu değer doğru tipte bir değişken üzerine atanabilir. Visual Studio uygulama geliştirme ortamı da (IDE –Integrated Development Environment-) buna destek verir. Bir metodu kullanmak isteyen programcının o metodu nasıl ele alacağı konusunda yardımcı olmak için metodun geri dönüş değerini gösterir.


```

using System;

namespace Metotlar
{
    class MetotKullanimi
    {
        static int Hesapla()
        {
            int a = 2;
            int b = 21;
            int sonuc = a + b;

            return sonuc;
        }

        public static void Main(string[] args)
        {
            Hesapla();
            int MetotKullanimi.Hesapla();
            Console.ReadLine();
        }
    }
}

```

.NET uygulama geliştirme ortamının metodun geri dönüş tipini kullanıcıya göstermesi

Metodun çağrıldığı yerde adı yazılırken veya yukarıdaki gibi yazıldıktan sonra imleç ile üzerine gelindiğinde, en başta metodun geri dönüş tipini görebiliriz. Buna göre eğer dönüş tipi **void** ise sadece metodu çağırırız. Dönüş tipinin void dışında herhangi bir veri tipi olması durumunda ise, uygun tipte bir değişken tanımlayıp, dönüş değerini bu değişken üzerine alırız. Bir metodun geri döndürebileceği veri tipi önceden tanımlı tiplerden biri olabileceği gibi, kendi yazdığımız bir tip de olabilir.

EĞİTİM :

DİZİLER VE METOTLAR

Bölüm :

**Diziler ve Koleksiyonlara
Giriş**

Konu :

Parametre Alan Metotlar



Microsoft Türkiye

Açık Akademi

Parametreler, metot içerisinde yapılacak işlerde kullanılmak için dışarıdan alınan değerlerdir. Metot oluşturulurken metot adından sonra gelen parantezlerin içerisinde bildirilir. Bu noktaya kadar, örneklerde parametre kullanılmamıştır.

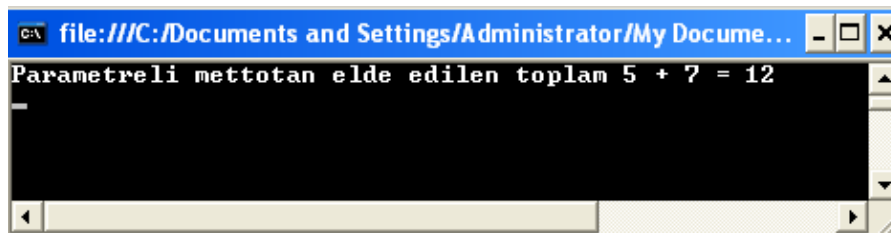
Her parametre, metoda tipi ve değişken adı ile bildirilir. Bu parametreler metot içerisinde lokal değişken gibi kullanılır. Metoda birden fazla parametre atanacağı zaman, parametrelerin virgül ile ayrılarak bildirilmesi gerekir.

```
static int Hesapla(int a,int b)
{
    int sonuc = a + b;
    return sonuc;    //Doğrudan "a + b" de return edilebilirdi.
}
```

Yukarıdaki metot, int tipinde iki tane parametre alıyor. Bu parametreler, a ve b adlarıyla metot içerisinde istenildiği gibi kullanılabilir. Örnekte bu iki değişken bir aritmetik işleme tabi tutuluyor. İşlem sonucu, başka bir değişkene ilk değer olarak atanıyor. Sonrasında da bu yeni değişkenin değeri, metodun dönüş değeri olarak kullanılıyor. Bu metodun kullanılışı aşağıdadır:

```
public static void Main(string[] args)
{
    int toplam = Hesapla(5, 7);
    //Artık metodun dönüş değeri toplam isimli değişken üzerinde
    Console.WriteLine("Parametrelili metottan elde edilen toplam 5 + 7 = " + toplam);
}
```

Hesapla() metodu yazılırken parantez içerisinde bildirilen iki tane int tipli değişken, kod bloğu içerisinde sanki varmış gibi kullanılır. Bu yapılırken değerleri ne olursa olsun metodun çalışacağı varsayılır. Değerlerin toplamı metodun geri dönüş değeri olarak başka bir değişkene atanır. Metot çağrılırken iki parametre değişkeninin alması istenen değerler, metodu çağırان tarafından verilir. Bu değerler, metodun kod bloğu içerisinde a ve b adıyla kullanılır. Gerekli hesaplamalar yapıp geri dönüş değeri elde edilir. Parametrelerin değerleri, metodun her çağrılışında değişik olabilir. Hesapla() metodu da her seferinde kendisine verilen farklı değerlerin toplamını geri döndürür. Bu metodun çıktısı aşağıdadır:



Parametre alan bir metodun çalıştırılması

Parametre alan metotların bir dönüş değeri olması zorunlu değildir.

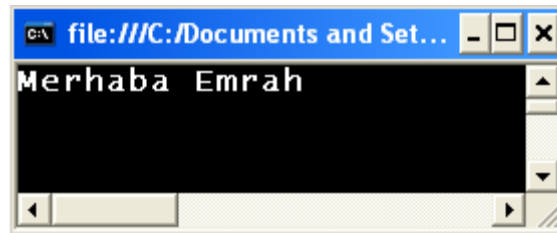
```
static void SelamVer(string ad)
{
    Console.WriteLine("Merhaba {0}",ad);
}
```

Yukarıdaki metot string tipinde bir parametre alır. Kod bloğu içerisinde parametrenin değerinin başına bir yazı getirilerek ekrana yazdırılır. Bu metodun kullanılışı aşağıdaki gibidir:

```
public static void Main(string[] args)
{
    SelamVer("Emrah");

    Console.ReadLine();
}
```

Bu metodun çalışmasıyla elde edilen çıktı aşağıdadır:

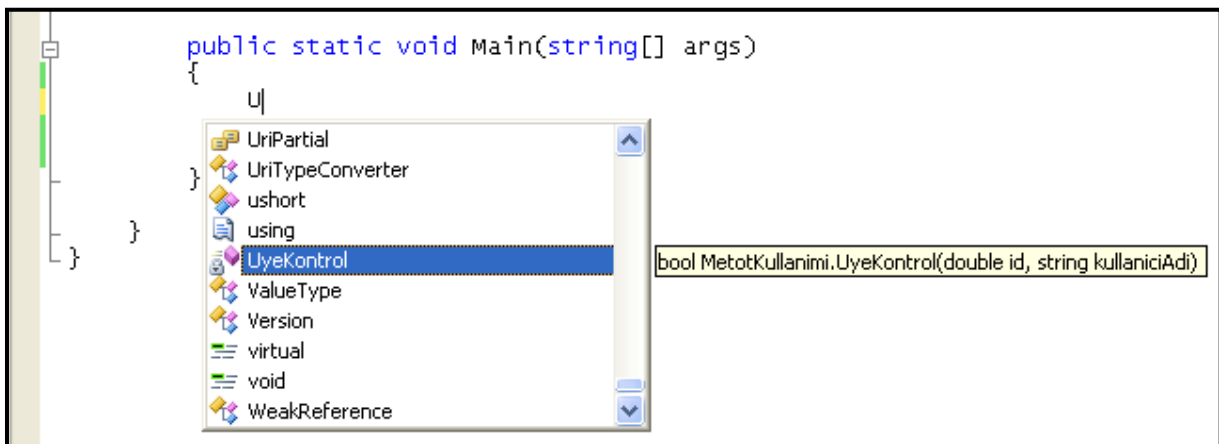


Dönüş değeri olmayan parametrik bir metot

Aşağıdaki metot, bu konunun son örneğidir.

```
static bool UyeKontrol(double id, string kullanicAdi)
{
    if ((id == 12345678) && (kullanicAdi == "emrhl"))
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

Bu metot, biri double diğeri string tipinde iki tane parametre alır. Bu parametreler ile bir kullanıcının web sitemizin kayıtlı bir üyesi olup olmadığının kontrol edildiğini varsayalım. Bu örnek siteye, sadece id'si **12345678** ve kullanicAdi **emrhl** olan kullanıcı kabul edilsin. Aşağıdaki şekilde bir metot çağırısı yapıp kullanılabildiğini düşünelim. Metodu çağırırken, metodun kaç tane ve hangi tipte parametre beklediği de intelli-sense özelliği sayesinde görülebilir:



Metot çağırılışında istenen parametrelerin IDE tarafından gösterilişi

Bu metodun geri dönüş tipi bool'dur. Alınan parametrelerin her ikisi de metot içerisinde kontrol edilen değerler ile aynı ise **true**; farklılık olması durumunda ise **false** değer dönecektir. Öyleyse metot çağrılıp, parametreleri verildikten sonra geri dönüş değeri bool tipte bir değişken üzerine alınıp kontrol edilmelidir.

```
public static void Main(string[] args)
{
    bool uyeMi = UyeKontrol(87654321, "lshrme");

    if (uyeMi == true)
    {
        Console.WriteLine("Login oldunuz...");
    }
    else
    {
        Console.WriteLine("Id ya da kullanıcı adı yanlış");
    }

    Console.ReadLine();
}
```

Metot, parametreleri verilip çağrılır ve geri dönüş değeri bir değişken üzerine alınır.

Verilen parametre değerleri metot içerisine yerleştirilip gerekli koşullar kontrol edilir. Koşullar sağlanmadığı için metottan geriye false döner. Main() içerisinde yapılan ise bu değişkenin değerine bağlı olarak kullanıcıya bir mesaj göstermektir.

EĞİTİM :

DİZİLER VE METOTLAR

Bölüm :

**Diziler ve Koleksiyonlara
Giriş**

Konu :

**Metotlara Parametre Aktarma
Yöntemleri**



Microsoft Türkiye

Açık Akademi

Değer Yolu ile Parametre Aktarmak

Metot parametreleri metoda aktarılırken varsayılan olarak değer yolu ile geçirilirler. Yani, metot çağrıldığında parametre olarak geçirilen bir değişkenin değerinin kendisi değil kopyası oluşturulup metoda özel olarak kullanılır. Metodun çalışması bitince de, bu kopya yok edilir. Bu durumu gözlemlemenin en güzel yolu aşağıdaki türde bir metottur.

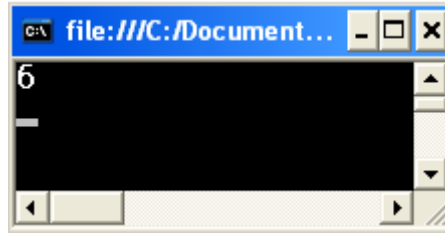
```
static void BirEkle(int x)
{
    x++;    //x'in değerini 1 arttırır.
}
```

BirEkle() metodu yardımıyla parametre olarak vereceğimiz bir değer gerçekten kendi değerinin mi artırılacağını yoksa bu artışın sadece metot içerisinde kalıp sonra yok mu edileceği incelenebilir.

```
public static void Main(string[] args)
{
    int k = 6;
    BirEkle(k);
    Console.WriteLine(k);

    Console.ReadLine();
}
```

Bu metodun çalıştırılmasıyla elde edilen çıktı aşağıdadır:



Parametrenin, metoda değer yolu ile geçirilişi

Main() içerisinde tanımlanan k isimli değişkene başlangıç değeri olarak 6 verilir. Ardından bu değişken BirEkle() metoduna parametre olarak gönderilir. Bu aşamada k değişkeninin değerinin bir kopyası (x) bellekte metot içerisinde oluşturulur. k değişkeninin orjinal değeri ise belleğin ayrı bir bölgesinde hala yerini ve değerini korur. Metot için oluşturulan bellek bölgesine, k değişkeninden elde edilen 6 değeri kopyalanır. Metot içerisinde kopya değişkenin değeri bir arttırılıp 7 yapılır. Metot sonlandığında ise bu değişken bellekten düşer. k değişkeni ise hala ilk tanımlandığı gibi 6 sayısal değerine sahiptir.

Referans Yolu ile Parametre Aktarmak

Metot çağrıldığında parametre olarak geçirilen bir değişkenin değerinin kendisi kullanılır. Değer yolundaki gibi bellekte yeni bir alan oluşturulmaz. Bir başka deyişle metoda, ilgili parametrenin bellekteki referansı aktarılmaktadır. Bir önceki örnek, referans yolu ile parametre geçirmeyi temsil etmek için kullanılabilir. Bir parametreyi metoda referans yolu ile geçirmek için hem metodun yazılışında hem de çağrılışında parametrenin ve değerin önüne **ref** anahtar kelimesi koyulur.

```

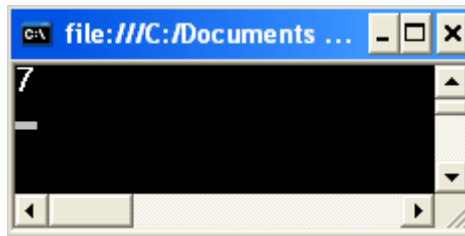
static void BirEkle(ref int x)
{
    x++;    //x'in deęerini 1 arttırır.
}

public static void Main(string[] args)
{
    int k = 6;
    BirEkle(ref k);
    Console.WriteLine(k);

    Console.ReadLine();
}

```

Main() içerisinde tanımlanan ve başlangıç değeri 6 olarak verilen k değişkeni, bellekteki adresi ile birlikte metoda parametre olarak geçirilir. Bu durum, parametrenin başına getirilen **ref** anahtar kelimesi ile sağlanır. Dolayısıyla kod bloęu içerisinde yapılan artış bu bellek bölgesini etkiler ve metod çağırısının ardından sorgulanan k'nın değeri 7 olarak elde edilir.



Parametrenin, metoda referans yolu ile geçirilişı

Output Yolu ile Parametre Aktarmak

Metoda parametreyi output yoluyla geçirmek, kavramsal olarak referans yolu ile parametre geçirmeye benzer. **ref** anahtar kelimesi yerine hem metod yazılışında, hem de metod çağırımında ilgili parametrelerin başına **out** anahtar kelimesi getirilir. **out** parametresinin tek farkı metoda **parametre olarak geçirilen değişkenin başlangıç değeri verilmesine gerek olmamasıdır**. **ref** anahtar kelimesi kullanıldığında ilgili değişken metoda aktarılmadan önce mutlaka ilk değer almalıdır.

```

static void KaresiniVer(int a,out int sonuc)
{
    sonuc = a * a;
}

public static void Main(string[] args)
{
    int kok = 5;
    int karesi;
    KaresiniVer(kok, out karesi);
    Console.WriteLine("5 in karesi : {0}",karesi);

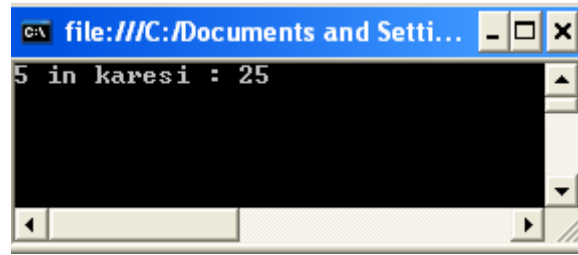
    Console.ReadLine();
}

```

Bu örnekte, KaresiniVer() metodu iki tane parametre kabul eder. Bu parametrelerden ilki karesi alınacak değerdir. Diğerisi ise, hesaplanan karenin değerini dışarıya (metodu çağırana) göndermek için kullanılacak olan

out parametresidir. Kod bloğunda **sonuc** isimli out parametresine **a** parametresinin karesi hesaplanıp atanır. Dikkat edilirse metot geriye hiçbir değer döndürmez gibi görünür. Fakat, aslında **out** parametresi sayesinde içeride hesaplanan bu parametrenin değeri dışarıya çıkarılır.

Bu bilgiler ışığında şunları söyleyebiliriz. Main() metodu içerisinde KaresiniVer() metodu çağrılır. 5 değeri verilen **kok** değişkeni ile başlangıç değeri verilmeyen ve out olarak işaretlenmiş **sonuc** değişkeni, bu metoda parametre olarak geçirilir. Burada gözlemlenmesi gereken, metoda parametre olarak geçirilirken out parametresinin bellekte yeri ayrılı olmasına rağmen henüz değerinin olmamasıdır. Metot içerisinde bu bellek alanına 5 değerinin karesi 25 atanır ve bu değer metot dışında da erişilebilir olduğu ekrana yazdırılıp görülür.



Parametrenin, metoda out yolu ile geçirilişi

Referans(ref) ve Output(out) yoluyla metotlara değer aktarma sayesinde, bir metottan geriye birden fazla değer döndürülebilir.

params Anahtar Kelimesinin Kullanımı

Bazı durumlarda metot yazılırken metodu çağıranın kaç tane parametre geçireceği önceden belli olmayabilir. Şu ana kadar öğrenilen bilgilerle bu ihtiyacı karşılayacak bir çözüm yoktur. Ancak, metotlara herhangi bir tipten bir dizi aktarılması halinde bu sorun kısmen de olsa aşılabılır. Bu iş için, metoda parametre olarak dizi alınması ve parametrenin önüne **params** anahtar kelimesinin getirilmesi yeterlidir:

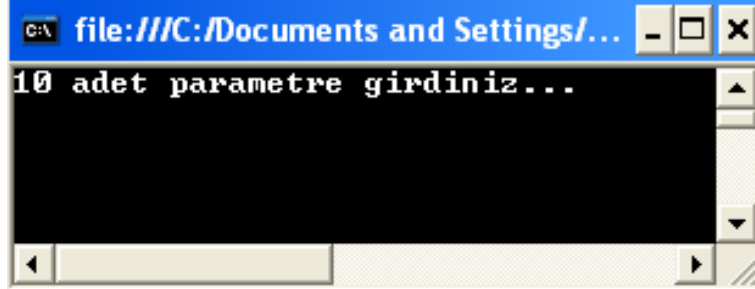
```
static void ParametreSayisiniVer(params int[] dizim)
{
    Console.WriteLine("{0} adet parametre girdiniz...",dizim.Length);
}
```

Yukarıdaki metodun çağrıldığı noktada, int tipinde, ihtiyaç duyulan sayıda parametre aktarımı gerçekleştirilebilir. Parametreler verilir uygulama çalıştırıldığında, parametre değerlerinden oluşan **dizim** adındaki dizi hafızada oluşturulur. Bu sayede metot içerisinde bu dizi ele alınabilir. **ParametreSayisiniVer()** metodu içerisinde girilen parametrelerin sayısı kullanıcıya söylenir.

```
public static void Main(string[] args)
{
    ParametreSayisiniVer (3, 4, 5, 6, 2, 5, 45, 4, 3, 2);

    Console.ReadLine();
}
```

Bu metodun çağırılmasıyla elde edilen çıktı aşağıdadır:



Parametreler, metoda geçirilirken params anahtar kelimesinin Kullanılışı

Şimdi sıra sizde; Siz de **params** anahtar kelimesi kullanarak, kullanıcının belirlediği sayıda tam sayı tipinde parametre alıp işlem yapan bir method yazabilirsiniz. Parametre olarak aldığı sayıların toplamalarını ve ortalamalarını bulan bir metot, güzel bir örnek olacaktır.

EĞİTİM :

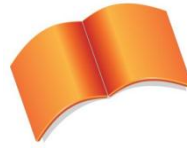
DİZİLER VE METOTLAR

Bölüm :

**Diziler ve Koleksiyonlara
Giriş**

Konu :

Not



Microsoft Türkiye

Açık Akademi

Metot İerisinde Kullanılan Deęiřkenler

Gerek metoda alınan parametreler, gerekse metot ierisinde oluřturulan deęiřkenler; metot, alıřma zamanında aęrıldıęı zaman belleęe alınırlar. Metodun alıřması sona erdięinde de bellekten dūřerler. Bu deęiřkenler sadece tanımlandıkları metot ierisinde kullanılabilirler. Dolayısıyla uygulamanın bařka herhangi bir yerinden bu deęiřkenlere eriřmek mūmkūn deęildir. Bu deęiřkenler, metot her aęrılıřında yeniden oluřturulur ve yok edilirler. Dolayısıyla bu deęiřkenler zerindeki deęerler metodun bir sonraki aęrılıřında elde edilemezler. Ayrıca metot ierisinde kullanılan bir deęiřken adı, bařka bir metot ierisinde de kullanılabilir. ūnkū her bir deęiřken kendi bloęuna zeldir.