Bölüm 01

Programlamanın Temelleri

C# Nedir?

Programın Evreleri

Program Yazmaya Giriş

Kaynak Programın Biçemi

Programa Açıklama Ekleme

Girdi/Çıktı İşlemleri

Veri Tipleri ve Değişken Kavramı

Metot Kavramı

Bellek Kullanımı ve Çöp Toplayıcı

Bu kitap hiç programlama bilmeyenler için yazılmıştır.

Bir programlama dilini öğrenmek, anadil öğrenmeye benzer. Çocukken, ana dilimizin sözdizimini (gramer) bilmeden, başkalarının yaptığı gibi kullanmaya başlarız; yani taklit ederiz. Ancak okula başladıktan sonra grameri adım adım öğreniriz. Bu derste genellikle bu yöntemi izleyeceğiz; yani C# dilini sistematik değil, pedagojik öğreneceğiz. Bir çok kavramı önce kullanacak, sonra onların esasını öğreneceğiz.

İlk bölümlerde söylenen kavramları esastan anlamanız beklenmiyor. Programların sözdizimlerinin (syntax - gramer) dayandığı temelleri iyi anlamasanız bile, bir çocuğun ana dilini öğrenmesi gibi, o kalıpları olduğu gibi kullanmaya çalışmalısınız. Yeni başlayanların bir dili öğrenmesinin en iyi yolu budur. Taklit etmek ve bolca tekrarlamak. Bu ders boyunca kavramları sık sık tekrarlayacağız ama her tekrarda o kavrama yeni bilgiler eklemiş olacağız.

Birinci Bölümde Nesne Yönelimli Programlama kavramına doğrudan girmek yerine, her programlama dilinde var olan ve programlamanın temelleri sayılan kavramlar, hiç programlama bilmeyenler için açıklanacaktır. Daha önce bir programlama dili öğrenmiş olanlar, bu bölümü okumadan atlayabilirler.

C# Nedir?

C# dilinin yaratıcısı olan Microsoft onu şöyle tanıtıyor:

C# basit, modern, nesne yönelimli, tip-korumalı ve C ile C++ dillerinden türetimiş bir dildir.

Bazılarına göre C# simgesinin sağındaki # simgesi, C++ simgesinin sağındaki ++ simgelerinin üst üste konmasıyla oluşturulmuştur.

Tabii, bu noktada Microsoft'un söylemediği bir gerçeği biz söylemeliyiz. Yukarıdaki sözlerde C# yerine çekinmeden *java* sözcüğünü koyabilirsiniz. Üstelik, java C# dan çok önce doğduğu için, o sıfatları fazlasıyla hakeder. Ancak, daha sonra yaratılan bir dilin, öncekilerin iyi yanlarını tamamen almış, iyi olmayan yanlarını iyileştirmiş olacağını tahmin edebiliriz. Bu tahmin bile , C# dilinin yeteneklerini anlatmak için yeterlidir.

C# simgesi İngilizce'de "Si-Şarp" diye okunur. C# kolay öğrenilir. C, C++ ve java dillerinin iyi özeliklerini almıştır. Kullanıcı dostu ve hızlı bir uygulama geliştirme (RAD :Rapid Application Development) aracı olduğu kabul edilir.

C# dilinin başlıca avantajları şunlardır:

Etkileşimli geliştirme aracıdır.

Windows ve Web uygulamaları için görsel tasarım sağlar.

Derlenen bir programdır (Scripting dili değildir).

Debug yapar.

C# dilinin başlıca nitelikleri:

Basitlik

Tip-korumalı

Önceki Sürümleri Destekleme

Nesne Yönelimli dillerin bütün niteliklerini taşır

Otomatik çöp toplayıcısı vardır

Oldukça esnek bir dildir

C# dilinin Başlıca Uygulamaları:

Class Library:- Başka uygulamalarda kullanılacak kütüphane sınıfları yaratır.

Console Application:- Satır komutu arayüzü için görsel C# uygulamaları yaratır.

Asp.Net Web Application:- Web kullanıcı arayüzü için görsel C# uygulamaları yaratır.

Asp.Net Web Service:- XML Web servisleri yaratır.

Asp.Net Mobile Web Application:- PDA, cep telefonları gibi taşınabilir cihazlar için uygulama programları yaratır.

Programlamanın Evreleri

Kullanılan dil ne olursa olsun, programlama eylemi şu evrelerden oluşur:

1. Kaynak programı yazma

Herhangi bir programlama dilinde text olarak yazılır. Kaynak program, kullandığı dilin sözdizimine (syntax) uymalıdır.

2. Derleme

Kaynak programı bilgisayarın anlayacağı bir ara dile dönüştürür. Ara dil kullanılan dile bağlıdır. Örneğin, C dilinde `obj` kodları, java dilinde `bytecode`, C# dilinde `IL`, vb adlar alır.

3. Aradil kodları yürütülebilir makina diline dönüştürülür

Programın yürütülebilir (çalışabilir, koşabilir - *executable*) olabilmesi için, kullanılan İşletim Sisteminin anladığı dile (makina dili) dönüşmesi gerekir. Windows İşletim Sisteminde yürütülebilir programların dosya adları `.exe` uzantısını alır.

Bu bağlamda bir konuyu daha açıklamakta yarar vardır. Çağdaş programcılıkta, her şey bir program içine yapılmaz. Daha önce yaratılmış, kullanılmaya hazır çok sayıda program öğeleri vardır. Bunlar, kullanılan dilin kütüphanesi gibidir. Programcı, kütüphaneden istediklerini alıp, kendi programına yerleştirebilir. Birleştirme yöntemi farklı olsa da her dilde bu olanak vardır. C# dilinin çok geniş bir kütüphanesi vardır. Onu kullanmayı gelecek bölümlerde öğreneceğiz.

Windows İşletim Sisteminde `.exe` uzantılı dosyaları kullanıcı yürütelebilir (koşturabilir).

Örneğin,

```
Deneme.cs
```

adlı bir C# kaynak programı bütün aşamalardan geçip makina diline dönüşünce

```
Deneme.exe
```

adını alır. Bu programı koşturmak için

```
Deneme
```

yazıp Enter tuşuna basmak yetecektir. Aynı işi, Windows'un görsel arayüzünde yapmak için, Windows Explorer ile Deneme.exe dosyasını bulup üstüne tıklamak yetecektir.

Program Yazmaya Giriş

Programlamaya yeni başlayanlar için, öğrenmenin en iyi yolu kitaplarda yazılan küçük programları usanmadan yazmak, derlemek ve çalıştırmaktır. Bu küçücük programlar, bilgisayarın yapabileceği bütün işleri size öğretecektir. Unutmayınız ki, büyük programlar bu küçücük programların birleşmesiyle oluşur. O nedenle, bu derste yazılacak küçük programların, programlamayı öğrenmenin biricik yolu olduğunu aklınızdan çıkarmayınız. Burada izlenecek aşamaların herhangi birisinde yapılacak yanlışlar, öğrenciyi bıktırmamalıdır. Aksine, programlamayı çoğunlukla yaptığımız yanlışlardan öğreniriz. O nedenle, yanlış yapmaktan korkmayınız. Üstelik yeni bir şey öğrenirken bilerek yanlışlar yapıp onun doğurduğu sonuçları görmek eğlenceli ve öğretici olabilir.

Bilgisayarınızda *Visual Studio 2008* kurulu olduğunu varsayıyoruz. Bunu kurmak istemeyenler C# derleyicisi ile de yetinebilirler. Her durumda C# derleyicisini kurulduğunu ve gerekli konfigürasyonların

yapıldığını varsayıyoruz. Eğer bu işler yapılmamışsa, önce kurulum işlemlerini yapmalısınız. Kurulumun nasıl yapılacağı ilgili web sayfalarından görülebilir.

Bu Bölümden sonra *Visual Studio 2008* bütünleşik uygulama geliştirme aracını kullanmaya başlayacağız. Bu aracın uygulama geliştirmede ne kadar yararlı ve kolay kullanılır olduğunu o zaman göreceksiniz. Ama bu aracı hemen kullanmaya başlamak, binanın tepesine asansörle çıkmak gibidir. Aşağı katlarda ne olup bittiğini anlamak için, en aşağıdan başlayıp medivenleri adım adım çıkmalıyız. Başlangıçta atacağımız bu zahmetli adımlar, bizim bilgisayarla nasıl iletişim kurduğumuzu anlamamızı sağlayacaktır.

O nedenle, ilk derslerde hiçbir görsel arayüz kullanmadan, işlerimizi doğrudan doğruya işletim sisteminin komutlarıyla göreceğiz. Böyle yapabilmek için, Windows'un Komut İstemi programını açacak ve Dos (Disk Operating System- Windows İşletim Sisteminin Anahtarları-) komutları yazacağız. Komut İstemi'ni aşağıdaki iki yoldan birisiyle açabilirsiniz:

Başlat -> Çalıştır sekmelerine basınca açılan pencerede Aç: ----- kutucuğuna cmd yazıp entere basınız.

Başlat -> Tüm Programlar -> Donatılar sekmelerinden sonra açılan alt pencereden Komut İstemi 'ne tıklayınız.

Bunlardan birisini yaptığınız zaman, ekranda siyah zemin üzerine beyaz yazılar yazılan Komut İstemi penceresinin açıldığını göreceksiniz. Bu pencereye istediğiniz dos komutunu yazabilir ya da yürütülebilir (executable) herhangi bir programı çalıştırabilirsiniz.

Şimdi yazdığımız programları içine kaydedeceğimiz bir dizin yaratalım. İstediğiniz sürücüde istediğiniz adla bir dizin ya da alt dizin yaratıp C# programlarınızı oraya kaydedebilirsiniz. Programlarımızı C: sürücüsünde yaratacağımız csprg dizini içine kaydetmek isteyelim. Bunun için şu komutları yazacağız:

```
md C:\csprg
cd C:\csprg
```

Bunlardan birincisi istediğimiz dizini yaratır, ikincisi ise o dizini etkin kılar. Basit bir deyişle, sistemin yazma/okuma kafası o dizin içine girer ve orada yazabilir ve okuyabilir. Dizine girdiğimizi, Komut İstemi ekranında beliren

```
C:\csprg>
```

görüngesinden anlarız. Bu simgeler daima Komut İstemi penceresinde görünecektir; etkin olan dizini gösterir. Dos komutlarını c:\csprg> etkin (prompt) simgesinden hemen sonra yazacağız. Şimdi program yazmamıza yardım edecek basit bir editör açalım. Komut İstemi penceresinde edit yazıp Enter tuşuna basınız:

```
edit
```

Eğer editör penceresi açılmazsa, sisteminiz edit.com dosyasını görecek biçimde ayarlı değildir. Bunu Ek_A da anlatıldığı gibi ayarlayabilirsiniz. Ama, şimdi zaman kaybetmemek için, edit.com dosyasının bulunduğu adresi tam yazarak, editörü açabiliriz:

```
C:\WINDOWS\system32\edit
```

Açılan editörde aşağıdaki programı yazıp Dosya sekmesinden Farklı Kaydet seçeneğine basınız. Açılan alt pencerede Dosya Adı yerine Program001.cs yazıp Enter tuşuna basınız. İsterseniz yazma ve kaydetme işini Notepad ile de yapabilirsiniz.

Program001.cs

```
class Program001
{
}
```

İpucu

C# kaynak programlarının dosya adlarının uzantısı hiç olmayabileceği gibi, istenen bir uzantı da yazılabilir. Ama, bir dosyanın C# kaynak dosyası olduğunu ilk bakışta anlayabilmemiz için .cs uzantısını yazmayı alışkanlık edinelim.

Program adı ile class adı aynı olmak zorunda değildir. Ama bu derste yazılan örneklerde, hangi sınıfın hangi programda olduğunu kolayca bulabilmek için, program adı ile sınıf adını aynı yapacağız. Bu yöntem *java* dilinde zorunludur.

C# nesne yönelimli bir dil olduğu için, hemen her şey sınıflar (nesneler) içine yazılır. Programda kullanacağımız değişkenler ve metotlar mutlaka bir ya da birden çok sınıf (nesne) içinde tanımlanmalıdır. Bunların nasıl olacağı, bu dersin asıl konusudur ve onları adım adım öğreneceğiz.

Yukarıda yazdığımız Program001.cs dosyası içinde tanımlanan Program001 sınıfının içi boştur. Dolayısıyla, bu sınıfın hiçbir işe yaramayacağı açıktır. Böyle olduğunu, C# derleyicimiz bize söyleyecektir. Bu programı derlemek için

```
csc Program001.cs
```

yazıp Enter tuşuna basınız. Eğer sisteminiz csc.exe ile adlandırılan C# derleyicisini görmüyorsa, derleyicinin bulunduğu adresi yazabilirsiniz.

```
C:\WINDOWS\Microsoft.NET\Framework\v3.5\csc Program001.cs
```

Derleyici size aşağıdaki iletiyi gönderecektir. (İletideki sürüm numaraları, sizin sistemde farklı olabilir.)

```
Microsoft (R) Visual C# .NET Compiler version 7.10.6001.4
for Microsoft (R) .NET Framework version 1.1.4322
Copyright (C) Microsoft Corporation 2001-2002. All rights reserved.

error CS5001: Program 'c:\csprg\Program001.exe' does not have an entry point defined
```

Bu iletinin ilk üç satırı, derleyiciye özgü olan ve telif haklarıyla ilgili bilgilerdir. Bizim için önem taşıyan ileti, error ile başlayan son satırdır. Bunu derin incelemeye gerek yoktur. Bize, Program001.cs dosyasını derleyip Program001.exe adlı yürütülebilir dosyayı yaratamadığını söylüyor. Bunu yapamayış nedenini, bir giriş noktası olmamasına bağlıyor. Bu hatayı da CS5001 numaralı hata olarak işaretliyor.

İpucu

Derleyici kaynak programda sözdizimi (syntax) hatası olduğunda programı derleyemez. Derleyici, derleyemediği her programda, sözdizimi yanlışlarını türlere ayırır ve her yanlış türüne bir kod numarası verir. Bu kod numaraları, programın düzeltilmesinde (debug) çok işe yararlar.

Ama, şimdilik bunları hiç düşünmeden, derleyicinin bizden istediği giriş noktasını oluşturmaya çalışalım.

C# programları bir `Main()` fonksiyonu tarafından eyleme geçirilir. Programın yapacağı bütün işler bu Main() fonksiyonu tarafından belirlenir. Buna programın giriş noktası diyoruz. Şimdi yukarıdaki programımıza `Main()` fonksiyonunu ekleyelim ve Program002.cs adıyla kaydedelim.

Program002.cs

```
class Program002
{
	static void Main()
	{
	}
}
```

Sonra programı derleyelim:

```
csc Program002.cs
```

Oley! Bu kez derleyicimiz hiç hata (error) iletisi göndermedi. Demek ki programı derledi ve yürütülebilir makina diline çevirdi. Gerçekten

```
dir
```

komutu verirseniz, c:\csprg dizini içinde Program002.exe adlı yürütülebilir bir program olduğunu göreceksiniz. Bu dosyayı koşturmak için

```
Program002
```

komutunu yazmanız yetecektir. Bunu yazdığınızda, program çalışır, ama siz ekranda bir şey göremezsiniz. Çünkü, programın içine ekrana çıkacak bir şey yazmadık. Şimdi bunu yapalım.

İpucu

Bir dili öğrenirken işin pedagojisi ile sistematiği asla paralel gitmez. Bu nedenle, bazen pedagojiden bazen sistematikten sapmak gerekir. Böyle durumlarda, görünmez uzaylı dostumuz Uzay bize ipucu verecektir. Şimdilik Uzay'ın bize verdiği ipuçlarını kullanalım. Zamanla, onları daha iyi kavrayacağız.

İpucu

Ekrana bir şey yazdırmak için `Write()` ya da `WriteLine()` fonksiyonları kullanılır. İkisi de aynı işi yapar, ancak birincisi isteneni yazdıktan sonra, yazdığı şeyin sonunda bekler, ikincisi isteneni yazdıktan sonra alttaki satırın başına geçer.

Şimdi bu ipucuna göre programımızı düzeltelim.

Program003.cs

```
class Program003
{
    static void Main()
    {
        WriteLine("Merhaba C# ")
    }
}
```

Bu programı kaydettikten sonra derleyelim:

```
csc Program003.cs
```

Derleyicimizin, bu kez,

```
Program003.cs(5,27): error CS1002: ; expected
```

hata iletisini verdiğini göreceğiz. Bu ileti kaynak programın 5-inci satırının 27-inci kolonunda (;) beklendiğini söylemektedir.

İpucu

C# dilinde her deyimin sonuna (;) konulur. CS1002 hata kodu, kaynak programda bir deyimin sonuna (;) konmadığında oluşur.

Derleyicinin hata iletisi kesin olmakla birlikte, ekran çözünürlüğüne bağlı olarak, hatanın oluştuğunu işaret ettiği yer (kolon) bize farklı görünebilir. O zaman, hatayı işaret edilen yere yakın yerlerde aramalıyız.

Şimdi eksik olan (;) yazarak programımızı düzeltelim (bu işleme 'debug' denir).

Program004.cs

```
class Program004
{
    static void Main()
    {
        WriteLine("Merhaba C# ");
    }
}
```

Bu programı Program004.cs adıyla kaydedelim ve derleyelim:

```
csc Program004.cs
```

Derleyicimiz başka bir hata iletisi gönderir:

> Program004.cs(5,3): error CS0103: The name 'WriteLine' does not exist in the class or namespace 'Program004'

Derleyicimiz bu kez, 5-inci satırın 3-üncü kolonunda hata bulmuştur. 'WriteLine' nın ne olduğunu Program004 içinde bulamamıştır.

İpucu

> C# dilinde her değişken ve her metot (fonksiyon) mutlaka bir sınıf içinde tanımlanır. `Write()` ve `WriteLine()` metotları `Console` sınıfı içinde tanımlıdırlar.
>
> Öyleyse, derleyicimize `WriteLine()` metodunu `Console` sınıfı içinde bulacağını söylemeliyiz. Bunu metodun önüne sınıfın adını koyarak yapıyoruz. Sınıf ile metot arasına (.) konulur. `Console.Writeline()` yazdığımızda, derleyici, `WriteLine()` metodunun `Console` sınıfı içinde olduğunu anlayacaktır.

Derleyicimizin uyarıları doğrultusunda kaynak programdaki eksiklerimizi tamamlıyor ve yanlışlarımızı düzeltiyoruz. Bu yaptığımız iş gerçek anlamda bir 'debug' işlemidir. Programcılıkta çok önemli olan bu işi erinmeden yapmalıyız. Yaptığımız yanlışlar bize çok şey öğretecektir.

Program005.cs

```
class Program005
{
     static void Main()
     {
          Console.WriteLine("Merhaba C# ");
     }
}
```

Bu programı Program005.cs adıyla kaydedelim ve derleyelim. Artık, derleyicinin her istediğini yaptığımıza göre, kaynak programımızın derlenebilmesini umut etmekteyiz.

```
csc Program004.cs
```

Heyhat! Derleyicimiz gene hata iletisi gönderiyor:

> Program005.cs(5,3): error CS0246: The type or namespace name 'Console' could not be found (are you missing a using directive or an assembly reference?)

hata gene 5-inci satırda oluşuyor. Derleyicimiz 'Console' un tanımını bulamıyor ve ipucu veriyor. Console'un bulunduğu yeri bir using directive ya da bir assembly reference olarak vermeyi unutmuş olabileceğimiz olasığı nı belirtiyor. Bu sorunu çözmek için Uzay'dan ipucu istemeliyiz.

Uzay dostumuz bize `WriteLine()` metodunun `Console` sınıfı içinde olduğunu, `Console` sınıfının da System namespace'i içinde olduğu ipucunu verdi. O halde, derleyicimizin istediği bilgileri ona verebiliriz.

Program006.cs

```
class Program006
{
     static void Main()
     {
          System.Console.WriteLine("Merhaba C# ");
     }
}
```

Artık son düzeltmeyi yaptığımızı umarak bu programı Program006.cs adıyla kaydedip derliyoruz:

```
csc Program006.cs
```

Oleeey! Derleyicimiz başka hata vermedi, programımız derlendi. Gerçekten

```
dir
```

komutunu yazarsak, C:\csprg dizininde Program006.exe yürütülebilir dosyasının yaratıldığını görebiliriz. Şimdi bu dosyayı koşturmak için,

```
Program006
```

yazıp Enter tuşuna basarsak, ekranda

```
Merhaba  C#
```

yazısını göreceğiz.

Buraya kadar altı adımda ekrana 'Merhaba C#' tümcesini yazdıran bir program yazdık. Bu bir tümce yerine bir roman ya da karmaşık matematiksel işlemlerden oluşan deyimler de olabilirdi. Uzun ya da kısa metinler veya basit ya da zor işlemler için yapacağımız iş hep budur. Bu biçimde yazacağımız küçük programlarla C# dilinin bütün hünerlerini öğreneceğiz. Büyük programların bu küçük programların uyumlu birleşmesinden oluştuğunu hiç aklımızdan çıkarmayalım. O nedenle, Visual Studio 2008 uygulama geliştirme aracını kullanmaya başlamadan, C# dilinin temellerini öğrenmeye devam edeceğiz.

Kaynak Programın Biçemi

Programlama dillerinde, belirli bir işi yapmak üzere yazılan sözdizimi (kod) parçasına deyim denir. Örneğin, System.Console.WriteLine("Merhaba C# "); bir deyimdir. C# dilinde bir deyimin bittiğini derleyiciye bildirmek için, o deyimin sonuna (;) konulur. Bir deyimdeki farklı sözcükler birbirlerinden bir boşluk karakteri ile ayrılır. Kaynak programda ardışık yazılan birden çok boşluk karakterleri tek boşluk olarak algılanır. Tablar, satırbaşları ve boş satırlar birer boş karakter olarak yorumlanır. Dolayısıyla, yukarıdaki programı

Program006a.cs

```
class Program006a { static void Main(){ System.Console.WriteLine("Merhaba C# ");}}
```

biçiminde ya da

Program006b.cs

```
class
Program006b
{
static


void
Main()


{
System.Console.WriteLine("Merhaba C# ");
}
}
```

biçiminde de yazabiliriz. Derleyici, kaynak programı derlerken birden çok tekrar eden boşluk karakterlerini, tab ve satırbaşlarını yok sayacaktır.

Buraya kadar C# diline ait bazı kavramları açıklamadan kullandık. Bu noktada basit açıklamalar yapmak yararlı olabilir.

Aduzayı (namespace)

C# kütüphanesinde çok sayıda metot ve değişken tanımları vardır. Metotların ve değişkenlerin her biri bir sınıf içindedir. Metotları ve değişkenleri içeren sınıfların sayıları da çok fazladır. Erişimi kolaylaştırmak için sınıflar gruplara ayrılmışlardır. Birbirleriyle ya da aynı konuyla ilişkili olan sınıflar bir grup içinde toplanır. Bu grupların her birine aduzayı (namespace) diye adlandırılan birer ad verilir. Başka bir deyişle, bir aduzayı değişken, fonksiyon, sınıf vb varlıkların adlarını içeren soyut bir ambardır (container).Örneğin `sistem` ile ilgili olan bütün sınıflar `System` aduzayı (namespace) içindedir. Bir aduzayında aynı adı taşıyan iki sınıf olamaz. Ama farklı aduzayları içinde aynı adı taşıyan sınıflar olabilir. Aduzay adları bu sınıfların karışmasını önler. Bir aduzayı (`namespace`) içerdiği varlıkları lojik olarak gruplar; yani grup üyelerinin fiziksel kayıt ortamında aynı yerde olmaları gerekmez.

Sınıflar (class)

Sınıflar (class) Nesne Yönelimli Programlamanın (Object Oriented Programming) temel taşlarıdır. Programın kullanacağı her değişken, her deyim, her metot (fonksiyon) mutlaka bir sınıf içinde tanımlanmalıdır. Bir sınıf içinde tanımlanan metotlar (fonksiyonlar) ve değişkenler o sınıfın `öğe`'leridir (member). Karışmamaları için, bir sınıfta aynı adı taşıyan iki öğe olamaz (aşkın öğeler kavramını ileride göreceğiz).

Uzay dostumuz bize `WriteLine()` metodunun `Console` sınıfı içinde olduğunu, `Console` sınıfının da System namespace'i içinde olduğu ipucunu verdi. O halde, derleyicimizin istediği bilgileri ona verebiliriz.

Bloklar

Kaynak program bir ya da bir çok sınıftan oluşur. Sınıfların her biri bir bloktur. Bir sınıf içindeki her metot (fonksiyon) bir bloktur. Örneğin, yukarıdaki programın tamamından oluşan

```
class Program006
{
}
```

bir bloktur. Benzer olarak,

```
static void Main()
{
}
```

metodu da bir bloktur. `Main(){ }` bloku, class Program006.cs { } blokunun içindedir. Bu tür bloklara iç-içe bloklar diyoruz. Bir blok içinde gerektiği kadar iç-içe bloklar oluşturabiliriz. Bir programda birden çok sınıf ve bir sınıf içinde birden çok blok olabilir. Bloklar, adına blok parantezleri diyeceğimiz { } simgeleri içine yazılır. Bazan Main(){ } blokunda olduğu gibi blok parantezlerinin önüne blok adı gelebilir. Her sınıf (class) ve her alt sınıf bir bloktur. Bir sınıf içinde yer alan her metot (fonksiyon) bir bloktur. İleride göreceğimiz if, case ve döngü yapılarının her birisi bir bloktur. Bir sınıf içinde birbirlerinden bağımsız bloklar seçkisiz sırada alt-alta yazılabilir. Ancak iç-içe olan bloklar, istenen işlem sırasını izleyecek uygun sırayla iç-içe konur. İç-içe blok yazılırken, en içten dışa doğru blokun başladığı ve bittiği yerleri belirten blok parantezlerinin ({ }) birbirlerini karşılaması sağlanmalıdır. Bu yapılmazsa derleme hatası doğar. İç-içe bloklarda, işlem öncelikleri en içten en dışa doğru sıralanır.

Derleyici kaynak programın biçemine (format) değil, sözdizimine bakar. Ama kaynak programı hem kendimiz hem de başka programcılar için kolay okunur biçimde yazmak iyi bir alışkanlıktır. İç-içe giren blokları birer tab içeriye almak, alt-alta yazılan bloklar arasına birer boş satır koymak, kaynak programın, programcı tarafından kolay okunup anlaşılmasını sağlar. Özellikle, uzun programlarda düzeltme (debug) ya da güncelleme (update) yaparken, kaynak programın yazılış biçemi işin kolay ya da zor yapılmasına neden olur.

Programa Açıklama Ekleme

Kaynak programımızın biçemi bilgisayar için değil, onu okuyan kişiler için önemlidir, demiştik. Büyük bir programın kaynağının kolay anlaşılır olmasını sağlamak programcının ahlâki sorumluluğundadır. Bunu sağlamak yalnızca programın biçemiyle olmaz. Büyük programlar için sınıfların, metotların, değişkenlerin işlevleri ayrıca açıklanmalıdır. Büyük yazılım firmaları programdaki her ayrıntıyı açıklayan döküman hazırlarlar. Böylece, yıllar sonra programcılar değişse bile, o dökümanlara bakılarak, kaynak program üzerinde değişiklikler, iyileştirmeler yapılabilir.

C# dilinde programa açıklama eklemek için iki yöntem kullanırız. Bu yöntemler C, C++ ve java gibi bir çok dilde de var olan yöntemlerdir.

Tek satır açıklaması

// simgesinden sonra satır sonuna kadar yazılanlar derleyici tarafından işlenmez. Bir satırın bütününü ya da bir satırın ortasından sonuna kadar olan kısmı açıklamaya koyabiliriz:

```
// Bu metot net ücretten gelir vergisini hesaplar
// Console.WriteLine("Merhaba dünya!");
return netÜcret;    // metodun değeri netÜcret 'tir
```

Birincide satırın tamamı bir açıklamadır. Örneğin, gelir vergisi hesaplayan bir metodun başladığı satırdan önceki satıra konulabilir, böylece metodun ne iş yaptığı her okuyan tarafından anlaşılır.

İkincide, genellikle program yazarken bazı deyimlerin programa konulup konulmamasının etkisini görmek için, programcının sık sık başvurduğu bir yöntemdir. Bir deyimin başına // simgelerini koyarak o deyimi derleyicinin görmesini engelleyebilirsiniz.

Üçüncüde, önce bir deyim yazılmış, deyimin bittiği yerde // simgesi konulmuştur. Derleyici deyimi görür ve işler, ama // simgesinden sonra satır sonuna kadar yazılanları görmez. Bu yöntemle, bir deyimin ne iş yaptığını kolayca açıklayabiliriz.

Çoklu satır açıklaması

Bazan yapacağımız açıklama tek satıra sığmayabilir. O zaman ardışık açıklama satırlarını /* */ simgeleri arasına alırız.

```
/*
Bu sınıf 20 Ağustos 2008 tarihinde
C#  ile Nesne Programlama
kitabının Onikinci Bölümü için yazılmıştır.
*/
```

Bu yöntem çok satırlı açıklamalar için kullanıldığı gibi, program yazarken ardışık satırlardan oluşan bir blokun, bir metodun, bir sınıfın programda olup olmamasının etkisini görmek için programcının geçici olarak başvurduğu önemli bir araçtır.

Girdi-çıktı İşlemleri

Kullanıcının bilgisayarla etkileşim içinde olabilmesi için, onunla iletişim kurabilmesi gerekir. Bunun tek yolu, bilgisayara `girdi (input)` göndermek ve ondan `çıktı (output)` almaktır. Hemen her dil bu

iş için çeşitli yöntemlere sahiptir. `Giriş/çıkış` birimleri dediğimiz birimler bu işe yarar. Örneğin, klavye, dosya, uzaktaki bilgisayar, ağ gibi araçlar bilgisayara girdi yapabilirler. Ekran, yazıcı, dosya, ses-aygıtları, ağ gibi araçlarla bilgisayardan çıktı alınabilir. Her gün sayısı ve niteliği değişen bu giriş-çıkış aygıtlarını kullanmakta C# çok hünerlidir. Onları ilerideki derslerde yeri geldikçe ele alacağız. Ama ilk derslerde çok kullanacağımız bazı kavramları tanımamız gerekiyor.

System aduzayı

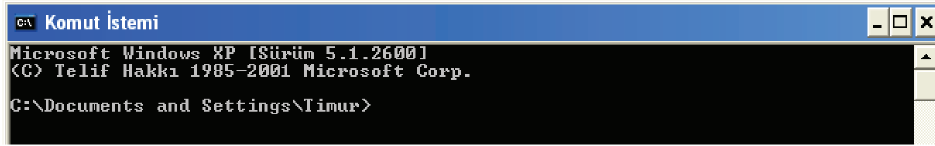
Bilgisayar sistemiyle ilgili sınıfları içeren bir aduzayıdır (`namespace`).

Console

`System` aduzayı içinde bir sınıftır. Konsol (console) uygulamaları için standart `giriş, çıkış` ve `hata` akışlarını belirler. `Console` sınıfından kalıtım (inherite) elde edilemez. Bildirimi şöyledir.

```
public static class Console
```

Bunların ne anlama geldiğini şimdilik merak etmenize gerek yoktur. Nesne yönelimli programlamanın alfabesi olan bu kavramlarla dersin ilerleyen evrelerinde karşılaşacak ve ne olduklarını kolayca anlayacağız. `Konsol (console)`, işletim sisteminin kullanıcıya açtığı bir etkileşim penceresidir. Bu pencere aracılığıyla kullanıcı ile sistem arasında iletişim kurulur. Kullanıcı klavyede yazdıklarını konsolda görebilir. Sistem kullanıcının konsola yazdığı komutları algılar ve ona yanıtları (tepki) gene konsola text tipi çıktı göndererek verir. Windows işletim sisteminde konsol, adına MS-DOS komut ekranı (command prompt) denilen şu penceredir.



Bu pencereden DOS komutları girilebilir ve sistem aynı pencereden kullanıcıya text tipi çıktılarla yanıt verir.

Konsol G/Ç Akım-yolları (Console I/O streams)

Bir konsol uygulaması başladığında, işletim sistemi kendiliğinden şu üç giriş/çıkış akım-yolunu konsola bağlar:

In
Out
Error

Kullanacağımız Windows işletim sisteminde standart giriş yolu klavye, standart çıkış yolu ekrandır. İstenirse bunlar değiştirilebilir. Örneğin klavyeden giriş yerine bir dosyadan giriş yapılabilir. Benzer olarak, ekran yerine yazıcıya veya bir dosyaya çıkış yapılabilir. Ama biz bu ders boyunca standart giriş birimi olarak klavyeyi, standart çıkış birimi olarak ekranı kullanacağız. Giriş `In`, çıkış `Out` ve hata `Error` akım-yollarıyla temsil edilir. Bunlar konsol sınıfında `girdi, çıktı` ve `hata` 'yı tutan değişkenlerdir. Bu değişkenlere nesne yönelimli programlama dillerinde, değişken demek yerine, `gendeğer` (property) denilir. Bazı kaynaklar "özellik" sözcüğünü kullanır. Bu kitapta `gendeğer` sözcüğünü tercih edeceğiz.

Yukarıdaki üç gendeğerin değerlerinin nasıl oluştuğunun ayrıntısına girmeden, konuyu basitleştirmek için şunu söyleyebiliriz. Bir konsol uygulaması başlayınca, kullanıcının konsola yazdıkları `In` akım-yoluna atanan değerdir. Sistem bu girdiyi algılar. Sistemin çıktıları `Out` akım-yoluna atanan değerdir. O değer konsola yazılır ve kullanıcıya iletilmiş olur. Bu işlemler sırasında bir hata oluşursa, o hata `Error` akım-yoluna atanır ve otomatik olarak konsola yazılır.

Girdi/Çıktı için Metotlar

C# dilinde girdi/çıktı işlemleri için `Console` sınıfına ait olan aşağıdaki metotları (fonksiyon) çok kullanacağız. O nedenle, onlar hakkında basit temel bilgileri edinmemiz gerekiyor.

Nesne Yönelimli Programlama dillerinde, fonksiyon kavramı matematikteki fonksiyon kavramından biraz daha geneldir. O nedenle, `fonksiyon` terimi yerine `metot` terimi kullanılır.

Write()

`Console` sınıfı içinde bir metottur (fonksiyon). Konsola çıktı gönderir. Sözdizimi:

```
Console.Write(parametre);
```

Bu metot parametre olarak C# dilindeki her veri tipini kabul eder.

WriteLine()

`Console` sınıfı içinde bir metottur (fonksiyon). Konsola çıktı gönderir. `Write()` metodu ile aynı işi yapar, ancak çıktıyı gönderdikten sonra ayrıca satırbaşı yapar (\r\n). Sözdizimi:

```
Console.WriteLine(parametre);
```

Read()

`Console` sınıfı içinde bir metottur . Standart giriş akım-yolundan gelen bir sonraki karakteri (henüz buffer'a girmeyen ilk karakteri) okur. Bu demek, eğer kullanıcı klavyeden giriş yapıyorsa, kullanıcının girdiği ilk karakteri okur. Arka arkaya `Read()` metodu kullanılıyorsa ~~yapılıyorsa~~, standart giriş akımından gelen karakterler geliş sırayla okunur. Sözdizimi:

```
Console.Read();
```

ReadLine()

`Console` sınıfı içinde bir metottur. Standart giriş akım-yolundan gelen sonraki satırı okur. Burada sonraki satır derken, henüz buffer'a girmeyen ilk satır kastedilmektedir. Arka arkaya `ReadLine()` metodu kullanılıyorsa, standart giriş akımından gelen satırlar geliş sırayla okunur. Sözdizimi:

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

Uyarı

Read() metodu bir karakter okur, `ReadLine()` metodu bir satır okur. Dolayısıyla, `Read()` metodunun değeri bir karakter, `ReadLine()` metodunun değeri bir `string`'dir. O nedenle, sayısal verileri `ReadLine()` metoduna `string` olarak okutur, sonra gerçek sayı tipine dönüştürürüz. Bu dönüşümü `Parse()` metodu ile yaparız..

Parametre

Bilgisayar programlarında, fonksiyonun bağlı olduğu değişkeni öteki değişkenlerden ayırmak için, onlara parametre denilir. Matematikte f(x) fonksiyonu için x değişkeni ne anlam taşıyorsa, bilgisayar fonksiyonları için parametre aynı anlamı taşır.

Parse()

Kullanıcının girdiği sayısal veri `ReadLine()` metodu tarafından string olarak okunur. Okunan bu veriyi olması gereken sayısal veri tipine dönüştürür. Sözdizimi:

```
string s;
```

```
int n;
```

değişken bildirimi yapılmış iken

```
s = Console.ReadLine(s);   n = int.Parse(s);
```

ya da `s` değişkenini kullanmadan

```
n = int.Parse( Console.ReadLine() );
```

yazılabilir.

.NET Framework C# veri tiplerini tanır ve onu kendi veri tipine dönüştürür. Dolayısıyla, C# veri tipleri yerine .NET teki karşılığını kullanabiliriz. Bu dönüşümü Veri Tipleri ve Değişkenler bölümde göreceğiz. Şimdilik, `int.Parse()` yerine `Int32.Parse()` yazabileceğimizi söylemekle yetinelim.

Veri Tipleri, Değişkenler ve Metotlar

Programlama dillerinde değişkenlerin ve metotların (fonksiyon, prosedür) işlevleri çok büyüktür. Değişkenler programın ham maddeleridir. Metotlar ise bu ham maddeleri işleyen araçlardır. Bunlarla ilgili ayrıntıları ilerideki derslere bırakıp, burada veri tipi, değişken ve metot kullanımına basit örnekler vereceğiz.

Program007.cs

```
class Program007
{
    static void Main()
    {
        int puan;
        puan = 19;
        System.Console.WriteLine(puan);
    }
}
```

Bu programı Program007.cs adıyla kaydedip derleyiniz ve koşturunuz. Çıktının

```
19
```

olduğunu göreceksiniz.

Bir programda farklı veri tipleriyle işlem yapmamız gerekebilir. Örneğin, tamsayılar, kesirli sayılar, karakterler (harfler ve klavyedeki diğer simgeler), metinler (string), mantıksal (boolean) değerler (doğru=true, yanlış=false) ilk aklımıza gelen farklı veri tipleridir. Bu farklı veri tiplerinin büyüklükleri (bellekte kaplayacakları yer) ve onlarla yapılabilecek işlemler birbirlerinden farklıdır. Örnekse, sayılarla dört işlem yapabiliriz, ama metinlerle yapamayız. O nedenle, C# ve başka bazı diller verileri tiplere ayırır. Değişken tanımlarken onun hangi tip veriyi tutacağını belirtir. Böylece, ana bellekte o değişkene yetecek bir yer ayırır ve o veri tipine uygun işlemlerin yapılmasına izin verir. Kabaca söylersek, değişken, ana bellekte belirli tipten bir veriyi tutması için ayrılan adrestir.

Aslında, C# dilinde her veri tipi bir sınıftır. Tersine olarak, her sınıf soyut bir veri tipidir. Bu konuyu ileride daha ayrıntılı inceleyeceğiz. Şimdilik, *int, char, string, bool vb.* temel veri tiplerini açıklamasız kullanacağız.

Şimdi bu programı satır satır inceleyelim. Bu basit işi yapmakla, bundan sonra yazacağımız programlardaki benzer ifadelerin ne iş yaptığını öğrenmiş olacağız.

1. Satır: Bu satır Program007 adlı bir sınıf tanımlamaya başlar.

2. Satır: { simgesi sınıf blokunun (sınıf gövdesi) başlangıcıdır.

3. Satır: `void Main()` ifadesi `Main()` metodunun (fonksiyon) belirtkesidir. Bu metodun adının Main, değerinin void (boş) olduğunu ve hiçbir parametreye (değişken) bağlı olmadığını belirtir. Bu satırın başındaki static deyiminin ne işe yaradığını dersin ilerleyen bölümlerinde öğreneceksiniz. Şimdilik size bir şey ifade etmese bile, şu basit açıklamayı yapacağız. Bir sınıfta `static` nitelemeli değişkenler ve metotlar, o sınıfa ait bir nesne yaratmadan kullanılabilirler.

4. Satır: { simgesi `Main()` metodunun blokunun (gövde) başlama yeridir.

5. Satır: Bu satırında yer alan int = puan; ifadesi bir deyimdir. Bu deyim iki iş yapar:

puan adıyla bir değişken tanımlar,

Bu değişkenin tutacağı verinin int (tamsayı) tipinden olacağını derleyiciye bildirir. Derleyici ana bellekte puan değişkenine verilecek tamsayı değerlerin sığabileceği büyüklükte bir yer ayırır. Bu yere puan değişkeninin bellekteki adresi denir. 'puan' adı o adresi işaret eden bir referanstır. Bazı dillerde buna pointer denilir. Ancak, C# , kullanıcıya pointer kullanma izni vermediği için, 'pointer' yerine 'referans' sözcüğü kullanılır.

6. Satır: Bu satırda yer alan puan = 19; ifadesi bir atama deyimidir. Önceki deyimde tanımlanan puan değişkenine 19 tamsayı değerini atar. Başka bir deyişle, puan değişkeninin bellekteki adresine 19 yazılır.

7. Satır: Bu satırda yer alan System.Console.WriteLine(puan); ifadesi puan değişkeninin değerini ekrana yazar.

8. Satır: } simgesi `Main()` metodunun bitiş yeridir.

9. Satır: } simgesi Program007 sınıfının bitiş yeridir.

Başlamışken, program üzerinde küçük değişiklikler yaparak, o değişikliklerin etkilerini görmeye çalışalım.

Atama Deyimi

Bir değişkene değer atamak demek, o değişkene ana bellekte ayrılan adrese bir değer girilmesi demektir.

```
int puan;
```

değişken bildirimi yapılmışken,

```
puan = 19;
```

deyimi bir atama eylemidir. Puan adlı değişkene 19 değerini verir. Atanan değer, değişken için ana bellekte ayrılan adrese yerleşir.

Yukarıdaki iki deyimi birleştirip tek deyim haline getiriniz.

```
int  puan = 19;
```

Programı bu biçimiyle derleyip koşturduğunuzda, aynı çıktıyı verdiğini göreceksiniz. Demek ki, değişken tanımını ve değer atamayı ayrı ayrı deyimlerle yapabileceğimiz gibi, değişken tanımını ve ilk değerini atamayı tek bir deyim ile de yapabiliriz. Ohalde,

```
int puan;
puan = 19;
```

deyimleri yerine

```
int  puan = 19;
```

deyimini yazabiliriz.

Şimdi programın yedinci satırını şöyle değiştirelim:

Program007a.cs

```
class Program007a
{
    static void Main()
    {
        int puan;
        puan = 19;
        System.Console.WriteLine(puan + 2);
    }
}
```

Bu programın çıktısı

```
21
```

dir. Demek ki WriteLine() metodumuz işlem yapabilmektedir.

Alıştırma

Bu metodun başka bazı hünerlerini görmek için, yedinci satır yerine, sırasıyla, aşağıdakileri yazarak programı her birisi için ayrı ayrı koşturunuz.

```
System.Console.WriteLine("Ankara Türkiye'nin başkentidir.");
System.Console.WriteLine(puan - 2);
System.Console.WriteLine(puan * 2);
System.Console.WriteLine(puan / 2);
System.Console.WriteLine(puan % 2);
System.Console.WriteLine(123 + 456);
System.Console.WriteLine(12*2 + 2 - 8/4);
```

Parametre ve Değişken

Matematik derslerinden anımsayınız. Bir fonksiyonu tanımlarken, onun değişkenlerini `f(x,y,z)` biçiminde yazarız. Bilgisayar programlarında da benzer işi yaparız. Fonksiyon adından sonra ( ) içine fonksiyonun bağlı olduğu değişkenleri yazarız. Bu değişkenleri, sınıf içinde bildirimi yapılan öteki değişkenlerden ayırmak için, 'değişken' yerine 'parametre' sözcüğünü kullanırız, demiştik. Buna ek olarak, yukarıdaki örneklerden görüyoruz ki, parametre bir işlem de olabiliyor. Bu durumuyla, parametre kavramı matematikteki değişken kavramına göre daha işlevseldir. Bu işlevsellik ilerideki konularımızda daha da artacaktır. Örneğin, yukarıdaki alıştırmalarda kullanılan WriteLine() metotlarının parametreleri, sırasıyla, şunlardır:

```
Ankara Türkiye'nin başkentidir.
puan -2
puan * 2
puan / 2
puan % 2
123 + 456
12*2 + 2 – 8/4
```

Bunların hepsinde parametre bir tanedir. Çıktılar, parametrelerin bellekteki değerleridir.

Metot Kavramı

Önemi nedeniyle, metot kavramını biraz daha açmamız gerekiyor. Gerektiğinde bir metotta birden çok parametre kullanabiliriz. Örneğin, aşağıdaki program `Hesap` adlı bir sınıf, sınıfın içinde `sonuç` adlı `int` tipinden bir değişken ile `Hesapla()` ve `Main()` metotlarını tanımlıyor.

Hesap.cs

```
using System;
class Hesap
{
    static int sonuç;
    static int Hesapla(int a, int b)
    {
        int x, y;
        x = 3 * a;
        y = b / 2;

        return x + y;
    }

    public static void Main()
    {
        int n;
        n = Hesapla(4, 8);
        sonuç = n;
        Console.WriteLine(sonuç);
    }
}
```

Bir programda yer alabilecek örnek öğeleri içerdiği için, programı satır satır çözümleyerek konuyu daha iyi anlatabiliriz. Burada yapacağımız basit açıklamaların gerisinde, C# dilinin sistematik yapısı yatmaktadır. O sistematik yapıyı ilerleyen derslerde adım adım açıklayacağız. Şimdi söyleyeceğimiz kavramları esastan anlamanız beklenmiyor. Programların sözdizimlerini (syntax -gramer) algılamaya çalışmanız ve onları şablon gibi kullanmanız yetecektir.

Baştan sona doğru açıklamaya başlayalım.

`using System;` deyimi `System` aduzayını (namespace) çağırıyor. Hemen her uygulamada bunu yapmamız gerekiyor.

`class Hesap` satırı Hesap adlı bir sınıf (class) bildirimini başlatıyor. Sınıfın bütün öğeleri en dıştaki {} bloku içinde yer alır. Bu bloka sınıf bloku veya sınıf gövdesi denilir. C# dilinde global değişken ve global metot yoktur. Bütün değişkenler ve metotlar sınıflar içinde tanımlanır. Bir sınıfın gövdesinde yer alan değişkenler ve metotlar o sınıfın öğeleridir (class member). Sınıfın öğelerine sınıf içinden doğrudan erişilebilir. Sınıfın öğelerine sınıf dışından erişmek için sınıf adı veya sınıfın bir nesnesi referans alınır.

`Static int sonuç;` deyimi sınıfa ait `sonuç` adlı int tipinden bir değişken tanımlar. Bu tür değişkenlere *sınıf değişkeni* ya da *sınıf öğesi* diyeceğiz. `sonuç` değişkeni `static` nitelemelidir. `static` nitelemeli değişkenleri ve metotları, sınıfa ait bir nesne yaratmadan kullanabileceğimizi söylemiştik.

```
    static int Hesapla(int a, int b)
    {
        int x, y;
        x = 3 * a;
        y = b / 2;

        return x + y;
```

```
    }
```

bloku `Hesapla()` adlı bir metot (fonksiyon) bildirimidir. Bir sınıf içinde bildirimi yapılan metot o sınıfın bir öğesidir. Metot adının sonuna gelen () parantezleri derleyiciye, yapılan bildirimin bir metot olduğunu bildirir. Bu parantezin içindeki `(int a, int b)` ifadeleri, metodun `int` tipinden `a` ve `b` adlı iki parametreye (değişken) bağlı olduğunu belirtir. Metot kullanılmak üzere çağrılırken `a` ve `b` parametrelerine istenen gerçek değerler atanır. Örneğin, `Main()` metodu, onu `Hesapla(4,8)` diye çağırmaktadır. Bu durumda `a=4` ve `b=8` ataması yapılıyor demektir. Bazan metodun parametresi olmaz. Parametreleri olmasa bile metot adının sonuna boş () parantezlerini koymak zorunludur. Metot adının önünde `int` yazılıdır. Bu metodun alacağı değerin `int` tipinden olacağını belirtir. Her metot bir değer almak zorundadır. Bu değer herhangi bir veri tipi ya da `void` (boş küme) olabilir.

```
int Hesapla(int a, int b)
```

ifadesi metodun adını, parametrelerini ve değer kümesini (alacağı değerin veri tipi) belirten bir ifadedir. Metoda ait gerekli her şeyi belirttiği için, bu ifadeye *metodun imzası* ya da *metodun belirtkesi* denilir.

Metot imzasını önündeki `static` nitelemesinin ne işe yaradığını yukarıda söylemiştik.

Metodun imzasından sonra gelen `{}` bloku *metot bloku* ya da *metot gövdesi* adını alır. Metoda ait değişkenler ve deyimler burada yer alır. Metot gövdesindeki

```
int x, y;
```

bildirimi x ve y adlı `int` tipinden iki değişken bildiriyor. Metot gövdesinde yer alan değişkenlere metodun yerel değişkenleri denilir. Yerel değişkenlere ancak metot içinden erişilir, dışarıdan erişilemez.

```
x = 3 * a;
y = b / 2;
```

deyimlerinin birincisi, metot çağrılırken `a` ya atanan değerin 3 katını `x` değişkenine atıyor. İkincisi ise, metot çağrılırken `b` ye atanan değerin yarısını `y` değişkenine atıyor.

```
return x + y;
```

deyimi yukarıda bulunan x ve y değerlerini toplayıp `Hesapla()` metodunun değeri olarak belirliyor. Dolayısıyla, `Hesapla(4,8)` çağrısında metodun değeri 16 olmaktadır. Değer kümesi void (boş küme) olmayan her metodun son deyimi `return` anahtar sözcüğü ile başlayıp değeri belirten bir ifade olmak zorundadır.

Bir programda herhangi bir sınıf içinde yer alan değişken ve metotlar kendi başlarına bir iş yapamazlar. Onların program içinde çağrılmaları gerekir. Bizim bu derste ağırlıklı olarak ele alacağımız konsol uygulamalarında, programın *giriş yeri* daima bir `Main()` metodu olacaktır. Dolayısıyla, her programın bir `Main()` metodu olacaktır. Main() metodu kendi sınıfında veya başka sınıflardaki değişken ve metotları çağırıp kullanabilir. Tabii, çağrıya olumlu yanıt verilebilmesi için, çağrılan öğenin erişim belirtecinin yeterli olması gerekir. *Erişim belirteçleri* 'ni ileride ele alacağız.

`Main()` metodunun gövdesindeki deyimlere bakalım.

```
    {
        int n;
        n = Hesapla(4, 8);
        sonuç = n;
        Console.WriteLine(sonuç);
    }
```

Birinci deyim, `int` tipinden `n` adlı bir yerel değişken tanımlıyor.

```
n = Hesapla(4, 8);
```

deyiminde eşitliğin sağ yanı, `Hesapla()` metodunu, parametrelerine `a=2` ve `b=8` değerlerini koyarak çağırıyor. Çağrılan `Hesapla()` metodu çalışıyor ve gövdesinde belirtilen işlemleri yaptıktan sonra bulduğu `x+y` sayısını `return` anahtar szcüğü ile değer olarak alıyor. Sonuçta, `Hesapla(4,8) = 16` değeri çıkıyor. Son olarak `n =Hesapla(4,8)` atama deyimi `n=16` atamasına denk bir iş yapmış oluyor.

Üçüncü satırdaki `sonuç = n` deyimi `n` yerel değişkeninin değerini `sonuç` adlı sınıf değişkenine aktarıyor. Son olarak,

```
Console.WriteLine(sonuç);
```

deyimi sonuç değişkenine atanan 16 değerini konsola yazıyor.

Write() ve WriteLine() Metotlarının Başka Hünerleri

Şimdiye kadar C# 'a bir şeyler yazdırmak için *System.Console* sınıfına ait Write() ve WriteLine() metotlarını kullandık. Şimdi onların yeni hünerlerini göreceğiz.

Program008.cs

```
class Program008
{
      static void Main()
            System.Console.WriteLine("Merhaba, ");
            System.Console.WriteLine("C# dersine hoş geldiniz. ");
}
```

Bu programın çıktısı

```
Merhaba,
C#  dersine hoş geldiniz.
```

biçiminde iki satırdan oluşur. Şimdi 4-üncü satırdaki WriteLine() yerine Write() metodunu yazalım:

Program008a.cs

```
class Program008a
{
      static void Main()
      {
            System.Console.Write("Merhaba, ");
            System.Console.WriteLine("C# dersine hoş geldiniz.");
      }
}
```

Bu programın çıktısı

```
Merhaba,  C#  dersine hoş geldiniz.
```

biçiminde tek satırdan oluşur. Bu ikisinden, daha önce de söylediğimiz bir ipucu çıkarabiliriz.

İpucu

> `Write()` metodu parametrelerin değerlerini ekrana yazar ve satırbaşı yapmaz. `WriteLine()` metodu ise parametrenin değerlerini ekrana yazar ve satırbaşı yapar.

Yukarıdaki programlarımız her koşmada ekrana bir tek ileti yazdılar; çünkü herbirinde WriteLine() metodunun bir tane parametresi vardı. Dolayısıyla, ekrana o parametrelerin değerleri yazıldı. Acaba programımız bir

koşmasında birden çok iletiyi yazamaz mı? Tahmin edeceğiniz gibi bu sorunun yanıtı 'evet' dir. Öyle olduğunu aşağıdaki örneklerden görebiliriz.

En kolay görünen ama en çok kaçınmamız gereken yöntem, her parametre için bir tane Write() ya da WriteLine() metodu kullanmaktır:

Program008b.cs

```
class Program008b
{
    static void Main()
    {
        int puan;
        puan = 19;
        System.Console.WriteLine(puan - 2);
        System.Console.WriteLine(puan * 2);
        System.Console.WriteLine(puan / 2);
        System.Console.WriteLine(puan % 2);
        System.Console.WriteLine(123 + 456);
        System.Console.WriteLine(12*2 + 2 - 8/4);
    }
}
```

Bu programın çıktısının

```
17
38
9
1
579
24
```

olduğunu göreceksiniz. Şimdi parametrelerin değerlerini aynı satıra yazdırmak için `WriteLine()` yerine `Write()` metodunu kullanalım.

Program008c.cs

```
class Program008c
{
    static void Main()
    {
        int puan;
        puan = 19;
        System.Console.Write(puan - 2);
        System.Console.Write(puan * 2);
        System.Console.Write(puan / 2);
        System.Console.Write(puan % 2);
        System.Console.Write(123 + 456);
        System.Console.Write(12 * 2 + 2 - 8 / 4);
```

```
    }
}
```

Bu programın çıktısı

```
17389157924
```

olur. Bu çıktı özünde doğru olmakla birlikte, parametrelerin değerleri bitişik yazıldığı için, okuyan birisinin çıktıyı doğru algılaması olanaksızdır. Bu sorunu çözmek için şu yöntemi izleyeceğiz:

Yer Tutucu Operatörü {}

{} parantezinin blokları belirlediğini söylemiştik. Şimdi onun başka bir işlevini göreceğiz. Önce aşağıdaki programın çoktılarına bakalım.

Program009.cs

```
class Program008d
{
    static void Main()
    {
        int puan;
        puan = 19;
        System.Console.Write("{0}  " , puan - 2);
        System.Console.Write("{0}  " , puan * 2);
        System.Console.Write("{0}  " , puan / 2);
        System.Console.Write("{0}  " , puan % 2);
        System.Console.Write("{0}  " , 123 + 456);
        System.Console.Write("{0}  " , 12*2 + 2 - 8/4);
    }
}
```

Bu programın çıktısı

```
17 38 9 1 579 24
```

olur. Bu çıktı özünde doğru olduğu gibi, okuyanı da yanıltmaz. ("{0} " , puan – 2) ifadesinde " " içindekiler bir metin olarak konsola yazılır. Bu metin içinde {0} simgesi yer tutucu operatördür. Metinden sonraki ilk değişkenin yazılacağı yeri gösterir. Yer tutucu operatörünün bir çok hünerini ilerleyen derslerde aşama aşama öğreneceğiz.

Şimdiye dek Write() ve WrireLine() metotlarının çıktıları tek bir parametre değeri oldu. Acaba birden çok parametre olunca, istenen parametrelerin değerleri yazdırılamaz mı? Bu sorunun yanıtının *'evet'* olduğunu aşağıdaki program göstermektedir.

Program0010.cs

```
class Program010
{
      static void Main()
      {
```

```
            System.Console.WriteLine("{0} {1}", 25 , 35 , "Merhaba");
      }
}
```

Bu programın çıktısı

```
25      35
```

biçiminde ilk iki parametreden oluşur. Bu ikisinden bir ipucu çıkarabiliriz.

İpucu

Parametreler soldan sağa doğru 0,1,2,... biçiminde numaralanır. Programlama dillerinin çoğu numaralama işlemini 1 den değil 0 dan başlatır. Biz de paramatreleri, soldan sağa doğru yazış sıramızla 0 dan başlayarak numaralayacağız. Buna göre,

```
"{0}"  ilk parametreyi (0-ıncı) yazar. Örneğimizde 0-ıncı parametre 25 dir.
"{1}"  1-inci parametreyi yazar. Örneğimizde 1-inci parametre 35 dir.
"{2}"  2-inci parametreyi yazar. Örneğimizde 2-inci parametre "Merhaba" metnidir.
...
"{n}"  n-inci parametreyi yazar.
```

Parametre değerlerinden istediklerimizi istediğimiz sırada yazdırabiliriz. Örneğin, yukarıdaki programda `{2}` yer tutucusu olmadığı için "Merhaba" parametresi çıktıya gitmemiştir. Aşağıdaki program önce 2-inci sonra 0-ıncı parametreleri yazdırır.

Program011.cs

```
class Program011
{
      static void Main()
      {
            System.Console.WriteLine("{2} {0} " , 25 , 35 , "Merhaba");
      }
}
```

Bu programın çıktısı şöyle olacaktır:

```
Merhaba 25
```

biçiminde olacaktır.

Write() ve WriteLine() metotlarımız her türlü aritmetik işlemi yapma becerisine sahiptirler. Aşağıdakine benzer programlar yazıp, sonuçlarını görünüz.

Program012.cs

```
class Program012
{
    static void Main()
    {
        System.Console.WriteLine(12 * 12 + 35 - 412);
    }
}
```

Değişken Kullanımına Örnekler

Değişkene atanan ilk değeri, program koşarken değiştirebiliriz. Bunu görmek için, programımızı aşağıdaki gibi değiştirelim.

Program013.cs

```
class Program013
{
    static void Main()
    {
        int puan = 19;
        System.Console.WriteLine(puan);
        puan = puan + 4;
        System.Console.WriteLine(puan);
    }
}
```

Bu programın çıktısı

```
19
23
```

olur. Neden böyle olduğunu kolayca anlayabilirsiniz. Kaynak programın 5-inci satırında puan değişkenine ilk değer olarak 19 atanmıştır. Bu değer geçerli iken 6-ıncı satırdaki WriteLine() metodu bu ilk değeri yazar. Ama arkasından gelen 7-inci satırdaki

```
puan = puan + 4 ;
```

deyimi, puan değişkeninin değerini 4 artırarak 23 yapmıştır. Dolayısıyla , 8-inci satırdaki WriteLine() metodumuz, o anda puan 'ın değeri olan 23 sayısını yazacaktır.

Şimdi, uzaylı dostumuz kulağımıza bir ipucu fısıldamaktadır.

İpucu

Yukarıdaki programın 8-inci satırındaki

```
puan = puan + 4 ;
```

atama deyimi

```
puan = 23;
```

atama deyimine denktir. Okuldaki matematik derslerinde yaptığımız işlemlere biraz aykırı görünen bu atama yöntemi, bilgisayar programlarında geçerlidir ve çok işe yarar. Özelikle, program koşarken bir değişkenin değeri, akışta oluşan koşullara göre değişebilir ve o değişimleri kaynak programda yazma olanağı olmayabilir.

Geçerlik Bölgesi (scope)

Şimdi değişkenlerin hangi bloklarda geçerli olabileceklerini araştıralım. Bunun için, puan değişkenini `Main()` blokundan alıp üst bloka koyalım. Programımız aşağıdaki biçimi alsın.

Program014.cs

```
class Program014
{
    int puan = 19;
    static void Main()
    {
        System.Console.WriteLine(puan);
    }
}
```

Yaptığımız değişiklik şudur: `Main()` blokundaki değişken tanımını ve değer atamayı, dış bloka aldık. `WriteLine()` metodu ise olduğu yerde, yani iç blokta kaldı. Bu programı derlemek istediğimizde, derleyici bize aşağıdaki iletiyi gönderir:

```
Program009.cs(6,28): error CS0120: An object reference is reququired for the static field, method, or property 'Program14.puan'
```

Bu iletiden anlamamız gereken şey, 6-ıncı satırda WriteLine() metodunun, parametre olarak yazdığımız `puan` değişkeninin değerine erişemediğidir. Çünkü, değişken adresini işaret eden bir işaretçi (referans, pointer) yoktur. Bu sorunu aşmak için, değişkenimize `static` nitelemesini vermek yetecektir. Neden böyle olduğunu ileride açıklayacağız.

Program014a.cs

```
class Program014a
{
    static int puan = 19;
    static void Main()
    {
        System.Console.WriteLine(puan);
    }
}
```

Bu programı koşturduğumuzda, çıktının

```
19
```

olduğunu göreceğiz. Demek ki, iç bloktaki WriteLine() metodu dış bloktaki değişkeni görebildi ve onun değerini yazdı.

Şimdi aynı değişkeni hem iç, hem de dış blokta tanımlayalım.

Program015.cs

```
class Program015
{
    static int puan = 19;
    static void Main()
    {
        int puan = 25;
        System.Console.WriteLine(puan);
    }
}
```

Programı derlediğimizde, derleyici aşağıdaki uyarıyı iletir, ama programı derler ve yürütülebilir dosyayı yaratır.

```
Program011.cs(3,14): warning CS0169: The private field 'Program015.puan' is never used
```

Gerçekten dir komutu ile Program015.exe dosyasının yaratıldığını görebilir ve

```
Program015
```

yazarak programı koşturabilirsiniz. Programın çıktısı

```
25
```

dir. Şimdi bunu biraz irdeleyelim. Dış blokta ve iç blokta ayrı ayrı iki tane puan değişkeni tanımladık. Dış blokta puan değişkenine 19, iç blokta ise 25 atadık. Program koştuğunda, WriteLine() metodu kendi bloku içindeki 25 değerini yazdı. Dıştaki değeri ihmal etti. Oysa, önceki programda, dış bloktaki değeri yazmıştı. Çünkü iç blokta aynı adlı bir değişken yoktu.

Bu noktada uzaylı dostumuza bakıyoruz. O bize şu açıklamayı yapıyor.

İpucu

İç blokta dış bloktaki değişken adları kullanılabilir. Ancak, aynı adı taşıdıklarında, iç bloktaki değişkenler öncelik alır.

Şimdi de olguyu tersine çevirelim. İç blokta tanımlı değişkenlerin dış blokta kullanılıp kullanılamayacağını öğrenmeye çalışalım. Bunun için `Main()` bloku içine bir blok koyalım ve değişkenimizi orada tanımlayalım.

Program016.cs

```
class Program016
{
    static void Main()
    {
        System.Console.WriteLine(puan);
        {
            int puan = 19;
        }
    }
}
```

Bu programı derlemeye kalkınca, aşağıdaki hata iletisini alırız.

```
error CS0103: The name 'puan' does not exist in the class or namespace 'Program016'
```

Bu ileti, 5-inci satırdaki `WriteLine()` metodunun puan değişkeninin tanımlandığı iç bloku göremediği anlamına gelir.

Şimdi uzaylı dostumuza danışmadan, kendimiz bir ipucu yazabiliriz.

Bazı pencereler dışarıdan bakılınca içerinin görünmemesi için özel bir maddeyle kaplanır. Dışarıdan bakınca camın içerisi görünmez, ama içeriden dışarısı olduğu gibi görünür. Bizim yazdığımız iç-içe bloklar bu özeliğe sahiptirler.

İpucu

İç-içe bloklar olduğunda, iç bloktan dış blok görünür, ama dış bloktan iç blok görünmez. Bu demektir ki, dış bloktaki değişkenlere iç bloktan erişilebilir, ama dış bloktan iç bloktaki değişkenlere erişilemez.

Dizi (array)

Dizi (array) kavramı hemen her dilde var olan önemli bir yapıdır. C# dilinde System.Array sınıfı arraylerle yapılabilecek bütün işleri belirler. Onu ayrı bir bölüm olarak ileride ele alacağız. Ancak o zamana kadar, yeri geldikçe diziler kullanmamız gerekecektir. O nedenle, bize oraya kadar yetecek basit bilgileri şimdi edinmemiz gerekiyor.

Array aynı veri tipinden çok sayıda değişkeni kolayca tanımlamaya ve o değişkenlere kolayca erişmeye olanak sağlayan bir yapıdır. Sözdizimi şöyledir:

```
tur[] ad =  {bileşen değerleri};
```

Örneğin,

```
int[]  puan = {68, 85, 45, 98, 74};                    (1)
```

bildirimi `int` türünden

```
puan[0], puan[1], puan[2],puan[3], puan[4]             (2)
```

`puan` adlı 5 tane değişken yaratır. Bunlara `puan` adlı array'in bileşenleri ya da öğeleri denilir. C# array'in bileşenlerini 0 dan başlayarak sırayla numaralar. (1) bildirimini şöyle de yapabiliriz:

```
int[]  puan  =   new int[5] ;
```

bildirimi puan adlı 5 bileşenli bir array yaratır. Bu arrayin bileleşenlerine değer atamak için, diğer değişkenler için yaptığımız atama deyimini kullanırız:

```
puan[0] = 68;
puan[1] = 85;
puan[2] = 45;
puan[3] = 98;
puan[4] = 74;
```

Array bildiriminde [ ] icindeki i = 0, 1, 2, 3, 4 sayılarına array'in indisleri (damga) denilir. Array içindeki her öğe kendisine ayrılan damga ile kesinlikle belirli bir değişkendir. Array sınıfının `Length` gendeğeri (property) arrayin uzunluğunu; yani kaç bileşeni (öğesi) olduğunu belirtir. Sözdizimi şöyledir:

```
puan.Length
```

Bunun değeri `puan` arrayimiz için 5 olur.

Array ile ilgili olarak söylediklerimiz şimdilik bize yetecektir.

For Döngüsü

Döngüler her programlama dilinde var olan önemli yapılardır. Onları ileride ayrı bir bölüm olarak ele alana kadar gerektiği yerde basit for döngüleri yazacağız. O nedenle, for döngüsünü basitçe açıklamamız gerekiyor.

Bir eylemin birden çok tekrar etmesini istiyorsak, o eylemi yaptıran kodu ard arda istenen sayıda yazabiliriz. Örneğin, yukarıdaki puan arrayinin bileşenlerinin sayısının 5 yerine 500 olduğunu düşünelim ve o bileşen değerlerini alt alta yazdırmak isteyelim. O zaman, `i` indisini `0` dan `499` 'a kadar değiştirerek

```
Console.WriteLine(puan[i]);
```

deyimini 500 kez yazmayı düşünebiliriz. Elbette, bu akıllıca bir yol değildir. Onun yerine, for döngüsü denilen şu yapıyı kullanırız:

```
for (int i = 0 ; i <500 ; i++)
{
    Console.WriteLine(puan[i]) ;
}
```

Bu döngüdeki kodları çözümleyelim:

`for` : döngünün başladığını bildirir.

(int i = 0 ; i <500 ; i++) : döngü sayacının int tipinden `i` değişkeni olduğunu ve ilk değerinin `0` dan başladığını; her adımda `1` artarak `500` den küçük kaldığı sürece `{}` döngü bloku içindeki deyim(ler)in tekrarlanacağını bildirir.

Bu söylediklerimizi bir araya getirerek aşağıdaki programı yazabiliriz.

ArrayYaz.cs

```
using System;

namespace BasitArray
{
    class birArray
    {
        static void Main(string[] args)
        {
            int[] puan;
            puan = new int[5];
            puan[0] = 68;
```

```
            puan[1] = 85;
            puan[2] = 45;
            puan[3] = 98;
            puan[4] = 74;

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

Bilgisayarda Bellek

Bilgisayarda çalışan bir programın kullandığı aduzayları, nesneler, değişkenler, deyimler, metotlar vb bilgisayarın belleğinde tutulur. Fiziksel yapılarına göre üç tür bellek vardır. Merkezi İşlem Birimi (CPU) içindeki register'ler, ana bellek (RAM) ve kayıt ortamları (hard disk, disk vb).

Registerler

Erişilmesi en hızlı olan bellek bölgeleridir; ama çok kısıtlıdırlar; CPU'nun mimarisine göre değişirler. Daha önemlisi, programcı register'lerin nasıl kullanılacağına karar veremez. Onları derleyici yönetir. Çok kullanılanları register'lerde tutarak hız kazanır.

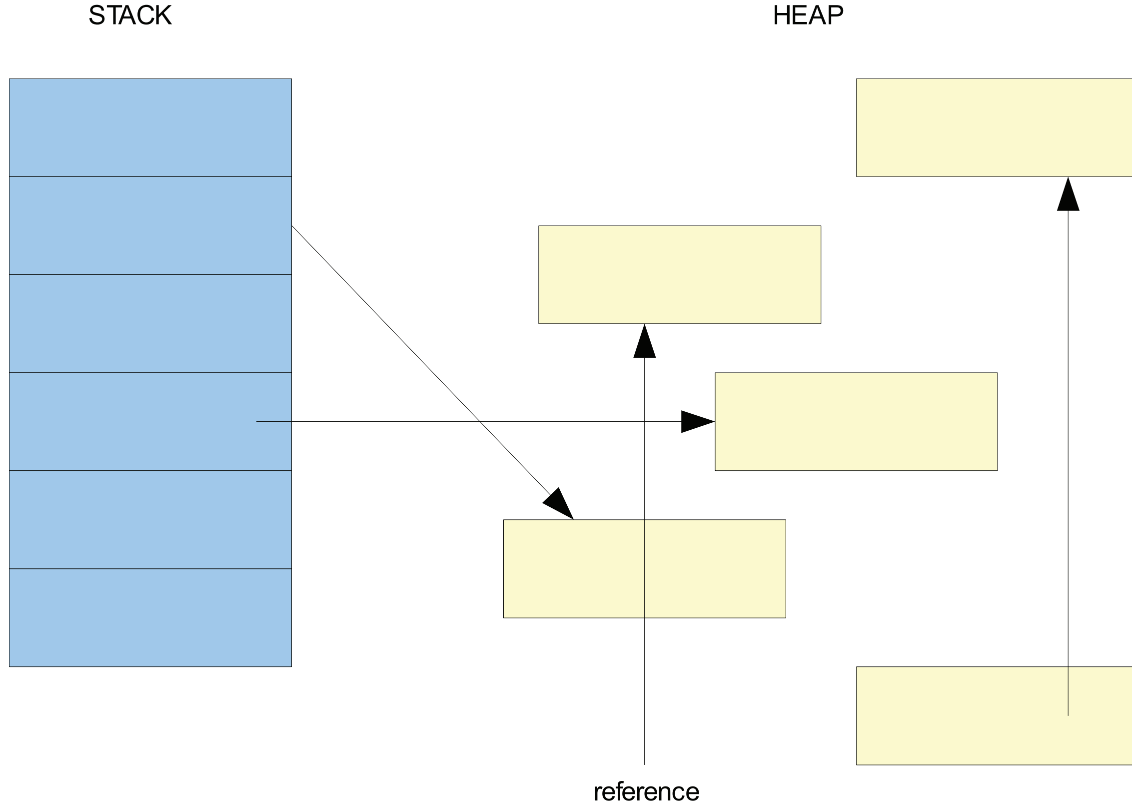
Kayıt ortamları

Program ana bellekten silindikten sonra gerektiğinde tekrar ulaşmak istediğimiz veriler, programlar ve benzerlerinin kalıcı olarak kaydedildiği ortamlardır. Hard disk, floppy disk, teyp, CD benzeri çok çeşidi vardır. Ayrıca, büyük programlar koşarken, ana bellek yetmediği zaman derleyici hard diskin bir bölümünü yardımcı bellek olarak kullanabilir. Swap space denilen bu alanı yönetmek derleyicinin ve sistemin tekelindedir.

Ana Bellek (RAM)

Program koşarken programın öğeleri ana bellekte belirli yerlerde tutulur. Çok ayrıntıya girmeden iki önemli bölgeyi söylemek gerekir: Stack ve Heap .

Stack : LIFO (Last Input First Output – son giren önce çıkar) adıyla bilinen yapıdır. Üst üste yığılmış kutuları andırır. Yeni gelen kutu en üste yerleşir, alınacak kutu en üstten alınır. RAM'de en hızlı erişim sağlanan bölgedir. C# dilinde bütün değer tipleri stack 'ta tutulur. `Stack` bir sınıftır ve stack ile ilgili bütün işleri yapmaya yetecek öğelere sahiptir.



Heap: C# dilinde bütün referans tiplerinin tutulduğu bellek bölgesidir; dolayısıyla nesneler `heap`'te tutulur. Bir sınıfa ait bir nesne yaratılınca, `Heap` içinde nesnenin bütün öğelerini içerecek bir bellek bloku ayrılır. Burası nesnenin adresidir. Bu adres, bu nesneyi işaret eden *referans* (işaretçi) tarafından bilinir. Şekildeki oklar referansları (pointer, işaretçi) göstermektedir.

Çöp Toplayıcı (Garbage Collector)

Java dilinde olduğu gibi, C# bir sınıftan yaratılan bir nesnenin işi bitince, ana bellekte ona ayrılan yeri boşaltır ve heap 'a ekler. Bu işin ayrıntısına girmeden, basit bir açıklama yapabiliriz. Bir nesne yaratılınca onu işaret eden bir referans (işaretçi) vardır. Örneğin,

```
{
string str = new string("Bu gün hava güzeldir.");
}
```

nesne kurucu deyimi ile yaratılan "Bu gün hava güzeldir." nesnesi ana bellekte bir adrese yerleşir. O adresi işaret eden referans `str` dir. Program kontrolu nesneyi içeren { } blokunu geçtiğinde `str` referansı yok olur. Ama "Bu gün hava güzeldir." nesnesi bellekteki yerini korumaya devam eder. C++ dilinde, işi biten nesneyi bellekten silmek, programcının görevidir. Programcı bunu doğru yapmadığı zaman, bellekte kendilerine asla erişilemeyen nesneler yer alır. Bu olgu C++ dilinin en büyük kusuru sayılır. Java ve ondan sonra gelen nesne yönelimli diller, Çöp Toplayıcı adını alan bir yöntemle, bellekte işleri bittiği için işaret edilmeden kalan nesneleri toplar ve çöpe atar; yani onların işgal ettikleri bellek adreslerini boşaltır ve o adresleri heap 'a ekler.

Dolayısıyla, C# programcısının, işi biten nesneyi silmek gibi bir yükümlülüğü yoktur; o işi Çöp Toplayıcı kendiliğinden yapacaktır. Ancak ileri düzeydeki programcılar, C++ dilindeki gibi pointer kullanmak, yarattıkları nesneleri işi bitince yoketmek isteyebilirler. C# buna kısıtlı izin verir. Ama bu konu, bu kitabın kapsamı dışındadır.

Alıştırmalar

1. Aşağıdaki program konsoldan girilen bir satırı string olarak okur ve onu konsola yazar.

```
using System;
namespace Bölüm01
{
    class InputOutput01
    {
        public static void Main()
        {
            string s;
            s = Console.ReadLine();
            Console.WriteLine(s);

        }
    }
}
```

Çıktı

Ankara başkenttir.

Ankara başkenttir.

Devam etmek için bir tuşa basın . . .

2. Aşağıdaki program konsoldan girilen karakterlerden yalnızca ilkini okur ve onu konsola yazar.

```
using System;
namespace Bölüm01
{
    class InputOutput02
    {
        public static void Main()
        {
            char ch;
            ch = (char)Console.Read();
            Console.WriteLine(ch);
        }
    }
}
```

Çıktı

765

7

Kullanıcı klavyeden 765 girdiği halde `Read()` metodu yalnızca ilk karakter olan 7 karakterini okumuştur. Burada şuna dikkat etmeliyiz. `Read()` metodu okuduğu karakteri, bizim gördüğümüz harf veya rakam biçimiyle değil, o karakterin `ASCII` koduyla algılar. Başka bir deyişle `Read()` metodu 7 karakterini okuduğunda buffer'a aldığı değer 7 değil, 7 rakamının ASCII kodu olan 55 dir. Dolayısıyla bu kodu 7 karakterine dönüştürmek için

```
ch = (char)Console.Read();
```

kapalı (implicit) dönüşüm operatörünü (char) kullanıyoruz. Bunu daha iyi anlamak için, yukarıdaki programı biraz değiştirelim.

Aşağıdaki program derlenemez. Çünkü, yalnızca sayısal veri tipleri arasında tip dönüşümü yapılabilir.

```
using System;
namespace Bölüm01
{
    class InputOutput03
    {
        public static void Main()
        {
            int ch;
            ch = (char)(Console.ReadLine());
            Console.WriteLine(ch);
        }
    }
}
```

Çıktı

```
Error    1        Cannot convert type 'string' to 'char' ...
```

Çıktıdan görüldüğü gibi, string'den char tipine açık (explicit) dönüşüm yapılamamaktadır.

3. Aşağıdaki program konsoldan girilen karakterlerden yalnızca ilkini okur ama (char) dönüşümü istenmediği için onu karaktere dönüştürüp konsola yazamaz.

```
using System;
namespace Bölüm01
{
    class InputOutput01
    {
        public static void Main()
        {
            char ch;
            ch = Console.Read();
            Console.WriteLine(ch);
        }
    }
}
```

Bu program sözdizimi (syntax) açısından yanlıştır; derleyici programı derlemez ve şu hata iletisini verir:

```
Error    1        Cannot implicitly convert type 'int' to 'char'. An explicit conversion exists (are
you missing a cast?)
```

4. Aşağıdaki program konsoldan girilen karakterlerden yalnızca ilk üçünü sırayla okur ve ekrana yazar.

```
using System;
namespace Bölüm01
{
    class InputOutput01
    {
```

```
        public static void Main()
        {
            char c1, c2, c3;
            c1 = (char)Console.Read();
            c2 = (char)Console.Read();
            c3 = (char)Console.Read();
            Console.WriteLine(c1);
            Console.WriteLine(c2);
            Console.WriteLine(c3);

        }
    }
}
```

Çıktı

```
a1b2c3d4e5f6
a
1
b
```