

**EĞİTİM :**

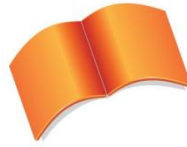
**MERHABA DÜNYA**

**Bölüm :**

**C# İle Yazılmış Bir  
Uygulamanın Temel Yapısı**

**Konu :**

**Bir C# Programının Yapısı**



Microsoft Türkiye

**Açık Akademi**

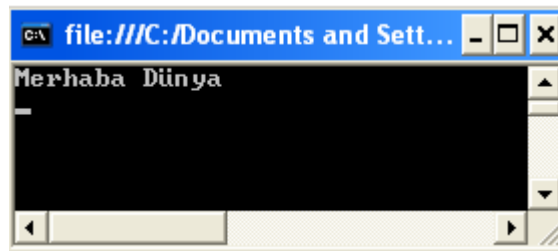
Bu bölüm çalışan basit bir örneği analiz ederek bir C# programının temel yapısını öğretmeyi hedeflemektedir. Bu bölümün içeriğinde ayrıca bazı temel giriş/çıkış operasyonlarını gerçekleştirmek için **Console** sınıfının nasıl kullanılacağı da yer almaktadır.

## Bir C# Programının Yapısı

Yeni bir programlama dili öğrenirken bir çok insanın yazdığı ilk program kaçınılmaz “Merhaba Dünya” dır. Aşağıdaki örnek kod bir C# programının gerekli bütün elemanlarını içerir:

```
//C# dosyaları, *.cs dosya uzantısı ile sona ererler.  
using System;  
  
class HelloWorldClass  
{  
    public static void Main(string[] args)  
    {  
        Console.WriteLine("Merhaba Dünya");  
        Console.ReadLine();  
    }  
}
```

Bu kod bloğu csc.exe ile komut satırından (csc.exe'nin kullanımı ilereleyen bölümlerde anlatılmaktadır) ya da Visual Studio ile derlendiğinde çıktı aşağıdaki gibi olur:



İlk uygulama

**EĞİTİM :**

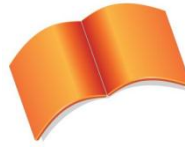
**MERHABA DÜNYA**

**Bölüm :**

**C# İle Yazılmış Bir  
Uygulamanın Temel Yapısı**

**Konu :**

**Sınıf Nedir?**



Microsoft Türkiye

**Açık Akademi**

## Sınıf (Class)

C# bütün programlama mantığının bir tip tanımlaması içerisinde olacağını söyler. Geliştirilen bir uygulama bir ya da daha fazla sınıf(class), yapı(struct) ve diğer tiplerden oluşan bir koleksiyon olur. (.NET dünyasında tip, şu kümenin üyelerinden birini anmak için kullanılan bir terimdir {Sınıf (class), yapı (structure), numaralandırıcı (enumeration), arayüz (interface), temsilci (delegate)}). Bu tiplerin arasında en temel olanı sınıftır. Sonraki bölümlerde ayrıntılı olarak görülecek olan sınıf; veri, bu verilerle ilişkide bulunabilen metotlar ve diğer üyeleri ile belli bir programlama görevini yerine getirmekle sorumludur.

“Merhaba Dünya” uygulamasındaki kodlara bakılacak olursa HelloClass adında tek bir sınıf olduğu görülür. Bu sınıf programa *class* anahtar kelimesi ile tanıtılırken sınıf adının ardından süslü parantez - { - gelir. Süslü parantez arasındaki her şey ilgili sınıfın parçasıdır.

Her bir C# sınıfı birer .cs dosyasına yazılabildiği gibi, bir .cs dosyasında birden fazla sınıf tanımlaması yapılabilir. Ayrıca C# 2.0 öncesinde bir sınıfı birden fazla fiziki dosyaya parçalayacak şekilde tasarlamak mümkün değilken, C# 2.0 ve sonrasında class anahtar kelimesinin önüne **partial** anahtar kelimesini koyarak bu mümkün olmaktadır. Aynı zamanda tüm sınıf tanımlamaları için .cs uzantılı dosya adı ile sınıf adı aynı olmak zorunda değildir; ancak genellikle değiştirilmez.

**EĞİTİM :**

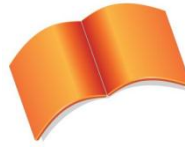
**MERHABA DÜNYA**

**Bölüm :**

**Visual Studio İle Tanışın**

**Konu :**

**Visual Studio Nedir?**



Microsoft Türkiye

**Açık Akademi**

## Microsoft Visual Studio

Microsoft Visual Studio; Microsoft teknolojileri üzerinde yazılım geliřtirmek için en fazla kullanılan yazılım geliřtirme aracıdır. İçerdiği araçlarla yazılım geliřtiricilere çok fazla kolaylık saęlayan Microsoft Visual Studio, yazılım geliřtirme deneyiminin çok üst seviyede yařanmasına olanak tanımaktadır. Yazılımcıların eli ayaęı olan Microsoft Visual Studio'ya biraz daha yakından göz atalım;

**EĞİTİM :**

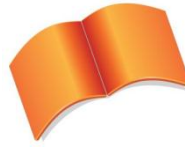
**MERHABA DÜNYA**

**Bölüm :**

**Visual Studio İle Tanışın**

**Konu :**

**Visual Studio İle Yeni Bir Proje  
Oluşturmak**

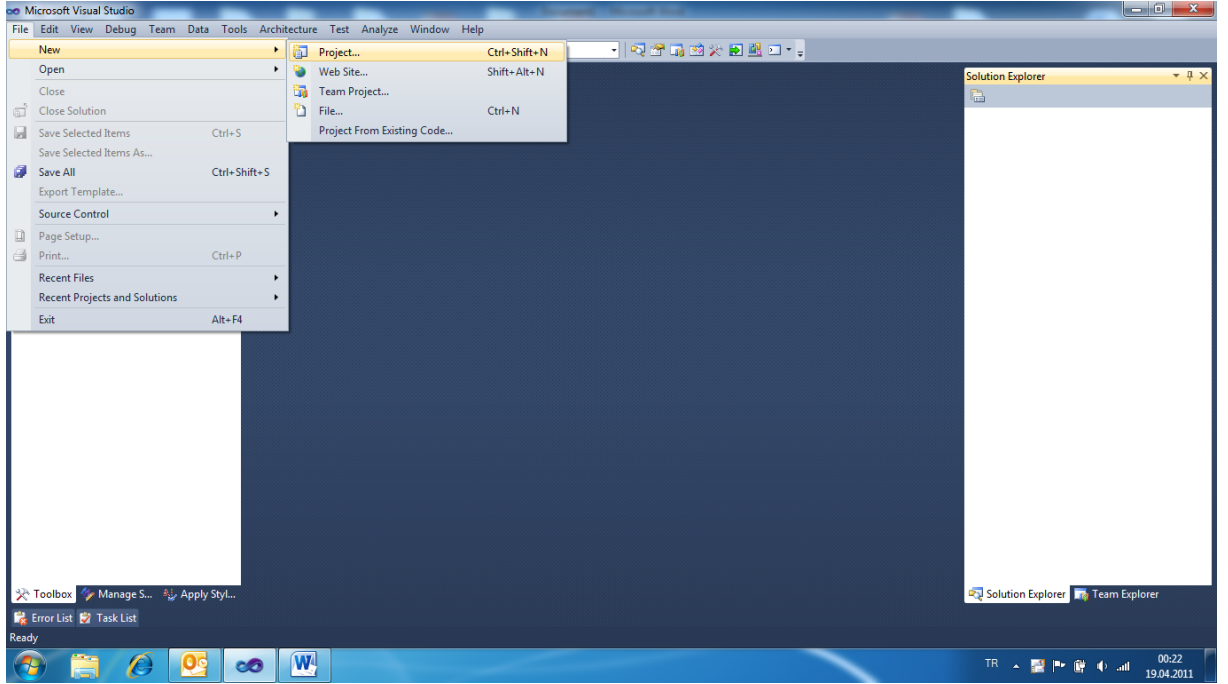


Microsoft Türkiye

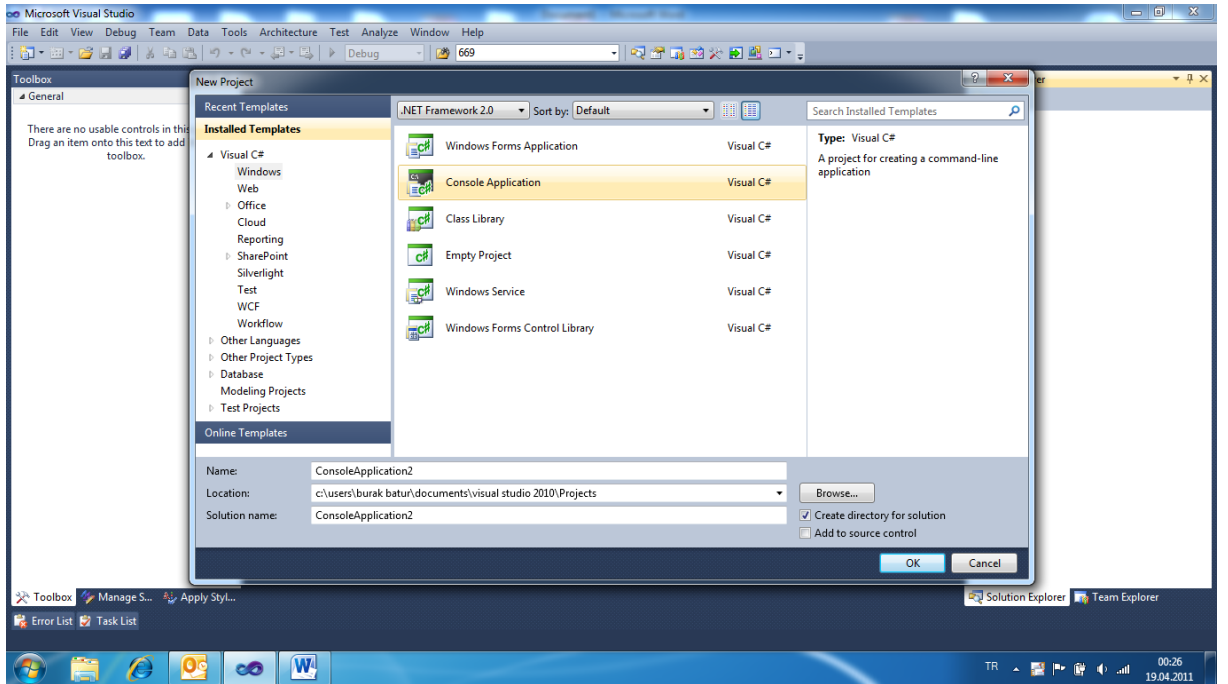
**Açık Akademi**

## Microsoft Visual Studio Üzerinde Yeni Proje Açmak

Yeni proje açmak için File menüsünden New seçeneğinin üzerine gelinir ve açılan menüden “Project” seçeneği seçilir.



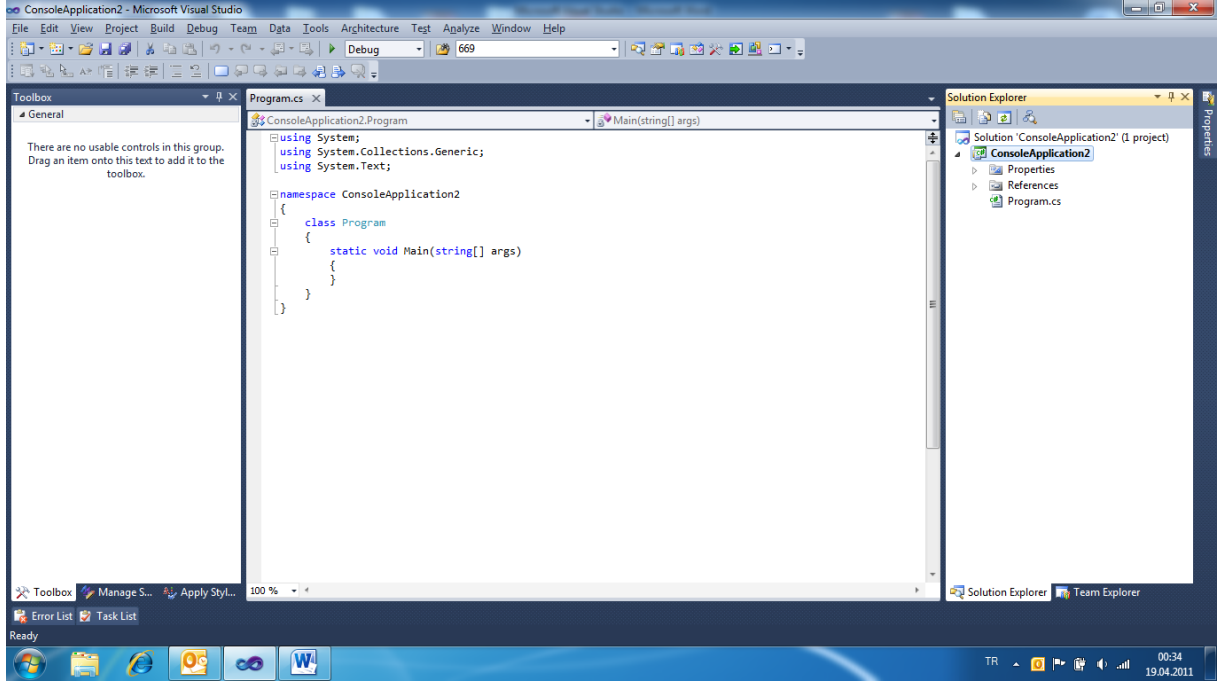
New prokekt menüsü karşınıza aşağıdaki ekranı çıkaracaktır. Yeni proje açmak istedinildiği Visual Studio'ya belirtildikten sonra bir de yeni açılacak olan projenin tipi belirtilmelidir. Sıradaki adımda projenin tipi belirtilmektedir.



Bu adımda Visual Studio üzerinde yüklü olan tüm proje şablonları listelenmektedir. Proje şablonları seçilen proje tipine göre gerekli olan minimum bileşenleri projeye dahil edip karşınıza çıkarmaktadır. Örneğin



bir masaüstü uygulaması geliştirmek istiyorsanız Windows Forms Application seçeneğini seçmelisiniz. Windows Forms Application size Word, Excel vs. gibi bir uygulama geliştirmek için gerekli alt yapıyı sağlar ve varsayılan olarak ana ekranınızı geliştirmenize olanak tanıyacak bir tane form getirir. Console Application seçeneği seçilirse DOS ekranı adını verdiğimiz komut satırında çalışan bir uygulama geliştirmeniz için gerekli bileşenler projeye dahil edilmiş olacaktır. Console Application seçeneğinde karşınıza form çıkarılmayacaktır çünkü komut satırı uygulamalarında forma ihtiyaç yoktur sadece çalışacak olan kodları barındıracak olan dosyalar hazırlanmış olarak karşınıza gelecektir. Bu bölümde kodların daha anlaşılabilmesi için tercih edilen örneklerde Console Application kullanılmaktadır. Proje şablonlarından Console Application seçilerek OK tuşuna basılır ve yeni proje açılır. Yeni proje ekranında aşağıdaki görünüm ile karşılaşılacaktır.



Yeni proje açıldıktan sonra karşınıza varsayılan olarak yukarıdaki görünüm geliyor olmalıdır. Bu görünümü kendinize göre özelleştirebilirsiniz. Font ayarları ile oynayıp fontların renginin, boyutunun vs. kendi zevkinize göre görünmesini sağlayabilirsiniz ya da burada bulunan ekranları kapatıp yenilerini görünür hale getirebilirsiniz.

**EĞİTİM :**

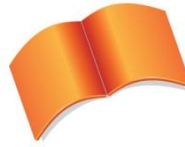
**MERHABA DÜNYA**

**Bölüm :**

**Visual Studio İle Tanışın**

**Konu :**

Visual Studio'da Solution  
Explorer Penceresi



Microsoft Türkiye

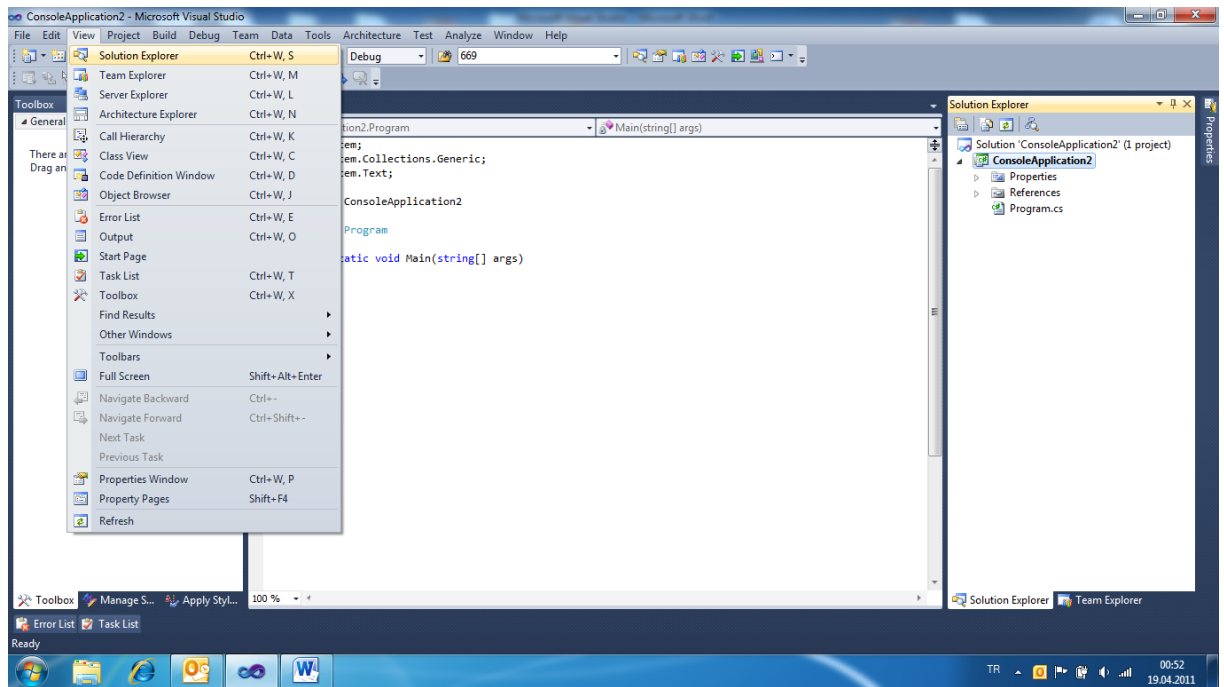
**Açık Akademi**

## Solution Explorer

Solution geliştirilmekte olan bir uygulamanın tüm bileşenlerinin bulunduğu yapıdır. Bir Solution içerisinde bir proje olabileceği gibi birden fazla proje de yer alabilir. Solution Explorer bir uygulamanın içerisinde bulunan tüm bileşenlerin listelenmiş olduğu bölümdür. Bu bölüm aracılığı ile uygulama içerisinde bulunan öğeleri görüp çift tıklayarak üzerinde çalışabilir, projeye yeni öğe ekleyebilir ya da var olan öğeyi silebilirsiniz. Solution Explorer'da en üstte Solution adı yer alır, Solution adının hemen altında şablonunu daha önceki adımda seçilen proje yer alır ve ağaç görünümünde bu proje içerisinde bulunan tüm bileşenler yer alır. Solution Explorer temel olarak uygulamaya göz atmanızı sağlar ve uygulama içindeki dokümanların yönetimi görevini sağlar.

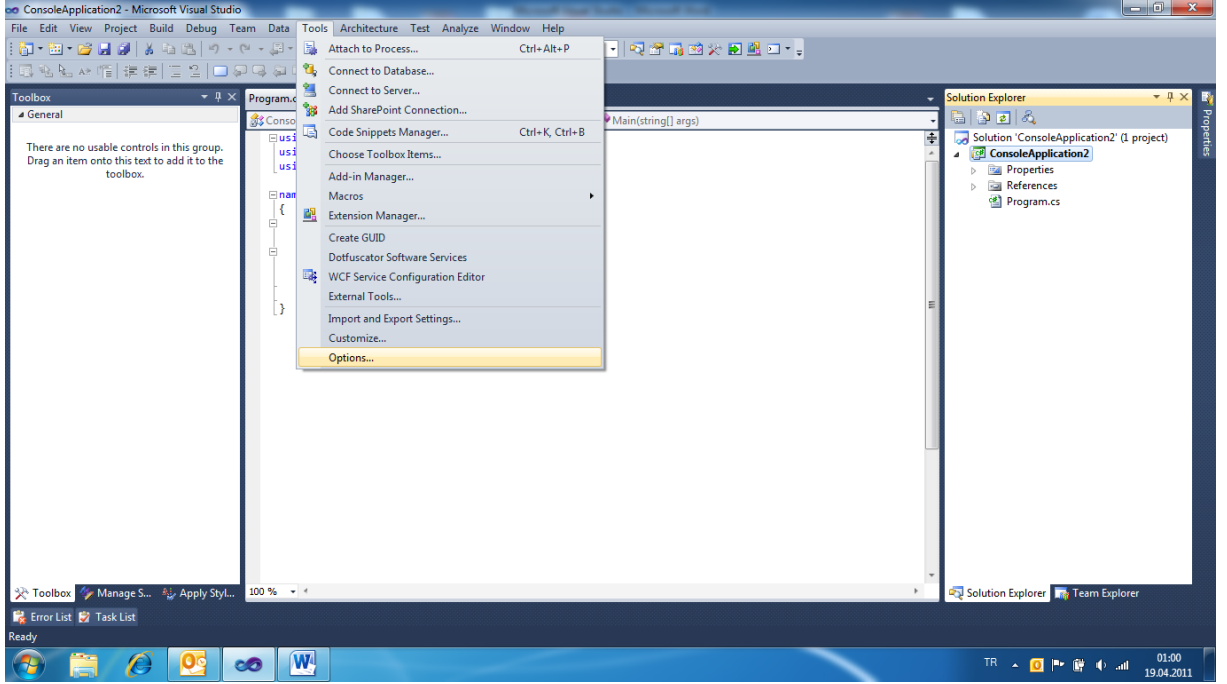
Visual Studio üzerinde yer alan View menüsü aracılığı ile karşınızda gördüğünüz bölümleri kapatıp açabilirsiniz. Örneğin kod geliştirirken sadece kod bölümü ile karşı karşıya kalmak için sol tarafta bulunan araç kutusu (ToolBox) ve sağ tarafta bulunan Solution Explorer bölümlerini üzerlerinde yer alan kapat butonu ile kapatabilirsiniz. Daha sonra bu bölümleri açmak için View menüsünden seçim yapabilirsiniz. Bölümlerin başlıklarına tıklayıp fareyi hareket ettirerek bölümleri Visual Studio ekranının istediğiniz bir bölümüne sabitleyebilirsiniz.

Not: ToolBox, Windows ya da Web tabanlı projelerde button, metin kutusu gibi görsel öğelerin bulunduğu bölümdür. İlgili proje şablonu seçildikten sonra bu bölümden istenilen bileşen form üzerine sürüklenip bırakılarak projeye dahil edilmiş olur. Eğitimin ilerleyen bölümlerinde ToolBox üzerinde yer alan bileşenlerden ve kullanımlarından bahsedilmektedir.

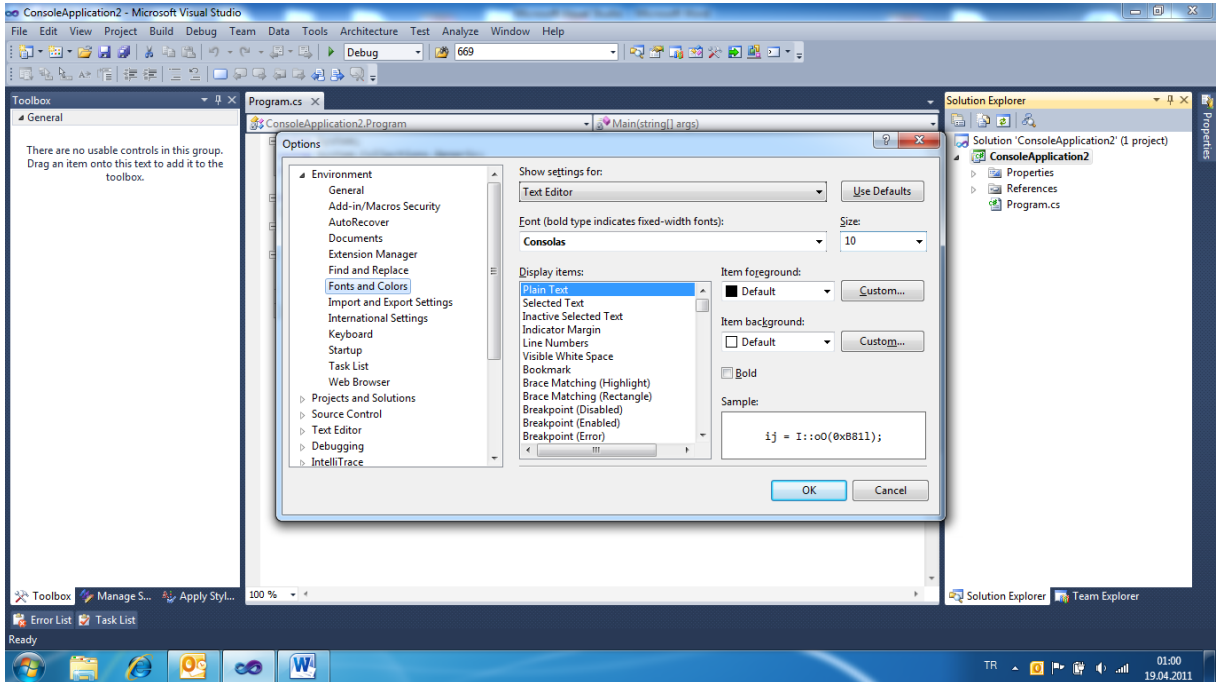


## Yazı Fontunun Kişiselleştirilmesi

Yazılım geliştiricinin iyi yazılımlar ortaya çıkarabilmesi için öncelikle kod yazmış olduğu editörün karşısında kendini iyi hissetmesi gerekmektedir. Yazılım geliştirici, yazılım geliştirdiği editörü kendi zevklerine ve çalışma alışkanlıklarına göre kişiselleştirebiliyor olmalıdır. Visual Studio size kişiselleştirme imkanları sağlamaktadır. Yazı tipini değiştirmek için **Tools** menüsünden **Options** seçilir.



Options bölümü Visual Studio'nun genel ayarlarının yer aldığı bölüme karşınıza getirecektir. Bu bölümden **Environment** sekmesindeki **Font and Colors** seçeneği üzerine gelinerek Visual Studio ekranlarında kullanılacak olan fontları kişiselleştirebilirsiniz.



**EĞİTİM :**

**MERHABA DÜNYA**

**Bölüm :**

**Visual Studio İle Tanışın**

**Konu :**

**Visual Studio'da Uygulama**

**Derleme ve Çalıştırma**



## Visual Studio Kullanarak Uygulamaların Derlenmesi

Visual Studio kullanarak uygulamaların derlenmesi için C# uygulamasını içeren proje açılarak **Build** menüsünden **Build Solution** tıklanır.

## Visual Studio Kullanarak Uygulamaların Çalıştırılması

Visual Studio kullanarak uygulamaların derlenmesi için **Debug** menüsünden **Start Debugging** (ya da Ctrl F5) ya da **Start Without Debugging** (ya da F5) tıklanır. Konsol uygulaması geliştirilirken Ctrl + F5 ile çalıştırılması, uygulama sonlandığında kullanıcı herhangi bir tuşa basana kadar pencerenin açık kalmasını sağlar. Bu iki çalıştırma yolunun farkı; ilkinde uygulama debug (hata ayıklama) modunda çalıştırılırken ikincisinde debug mod kapalı olarak çalıştırılmasıdır.

## Derleme Zamanı Hataları

Hem komut satırından hem de Visual Studio 2005 kullanarak uygulama derlendiğinde csc.exe, sözdizimi ya da semantik (anlam) bir hata yakalarsa bunu raporlar. Eğer komut satırından derlenmişse ekrana yazdırılan mesaj ile hatanın dosyadaki satır sayısı ve karakter pozisyonu bildirilir. Eğer derleme için Visual Studio kullanılmışsa Hata Listesi (ErrorList) penceresi bütün hatalı satırları işaret edip hata mesajını bildirir; hataya çift tıklanmasıyla da uygulama ortamı bizi ilgili satıra götürecektir. Uygulama derlendiğinde artık hata alınmıyorsa derleme zamanı hatalarından arınmışız demektir; uygulamanın .exe'si çalıştırılabilir.

	Uygulama derlenmeden çalıştırılırsa, otomatik olarak önce derlenir, ardından hata yoksa çalıştırılır.
--	---

## Çalışma Zamanı Hataları

Uygulama çalıştırılabilir dosyasının (\*.exe) sorunsuz olarak üretilmesinin ardından çalıştırılır ve eğer çalışma sırasında bir hata oluşursa buna **çalışma zamanı hatası (exception)** denir. Uygulama geliştirici olarak çalışma zamanında oluşacak hatalara karşı kod yazmamışsak bütün programlama ortamları için uygulama ekranında hatayla ilgili bilgilerin yer aldığı bir ekranla karşılaşılır. Uygulamadan faydalanan kullanıcının böyle bir ekranla karşılaşması istenmeyeceği için (en azından anlamlı bir hata mesajı ile karşılaşmasını isteyeceğimiz) bu hataları ele alan kodların yazılmış olması gerekir. (Bu konu "Çalışma zamanı hatalarını ele alma" başlığı altında ileriki konularda detaylı olarak incelenecektir).

## Visual Studio Hata Ayıklayıcısı Yardımıyla Uygulamanın İzlenmesi

Uygulamaya kesme noktaları (break point) koymak ve kullanılan değişkenlerin çalışma zamanı değerlerini incelemek,takip etmek için Visual Studio Hata Ayıklayıcısı'ndan (Visual Studio Debugger) faydalanılabilir.

Eğer bir uygulama satır satır ilerlenerek çalıştırılmak istenirse **Debug** menüsünden **StepOver** kliklenerek çalıştırılır ve yine aynı şekilde uygulama sonlanana kadar ilerlenebilir; satırlar arası ilerlerken uygulamanın nerelere dallanıp çalıştığı takip edilebilir ve kullanılan değişkenlerin değerleri incelenebilir.

Uygulamaya kesme noktaları koymak için kod yazdığımız C# dosyasında (\*.cs) herhangi bir satıra sağ tıklayıp **BreakPoint** seçeneğinden **Insert BreakPoint** tıklanır. Aynı zamanda sol kenar boşluğuna tıklanarak da koyulabilir. Kaldırmak için yine aynı satır üzerinden **BreakPoint** seçeneğinden **Delete BreakPoint** tıklanır. Aynı zamanda sol kenar boşluğunda çıkmış olan işarete tekrar tıklanarak da kaldırılabilir. Uygulama debug modda çalıştırıldığı zaman (F5 ile) herhangi bir satır ya da satırlara koyulan kesme noktasına kadar çalışıp o satırda bekleyecektir. Kullanılan değişkenlerin o anki değerlerinin incelenme ihtiyacı duyulduğunda bu faydalı olacaktır. Daha sonra uygulama ister adım adım (F10 ya da F11 ile) ister normal şekilde (F5) çalıştırılmaya devam edilir.

	F10 (Step Over) ile adım adım hata ayıklama yapılırken eğer kullanılan bir metod vb... varsa onların içerisine girilmezken F11 (Step Into) kullanıldığında her birinin içerisine girilir.
--	---

**EĞİTİM :**

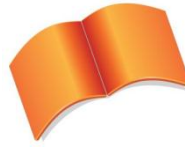
**MERHABA DÜNYA**

**Bölüm :**

**Main Metodu**

**Konu :**

**Main Metodu**



Microsoft Türkiye

**Açık Akademi**

## Main Metodu

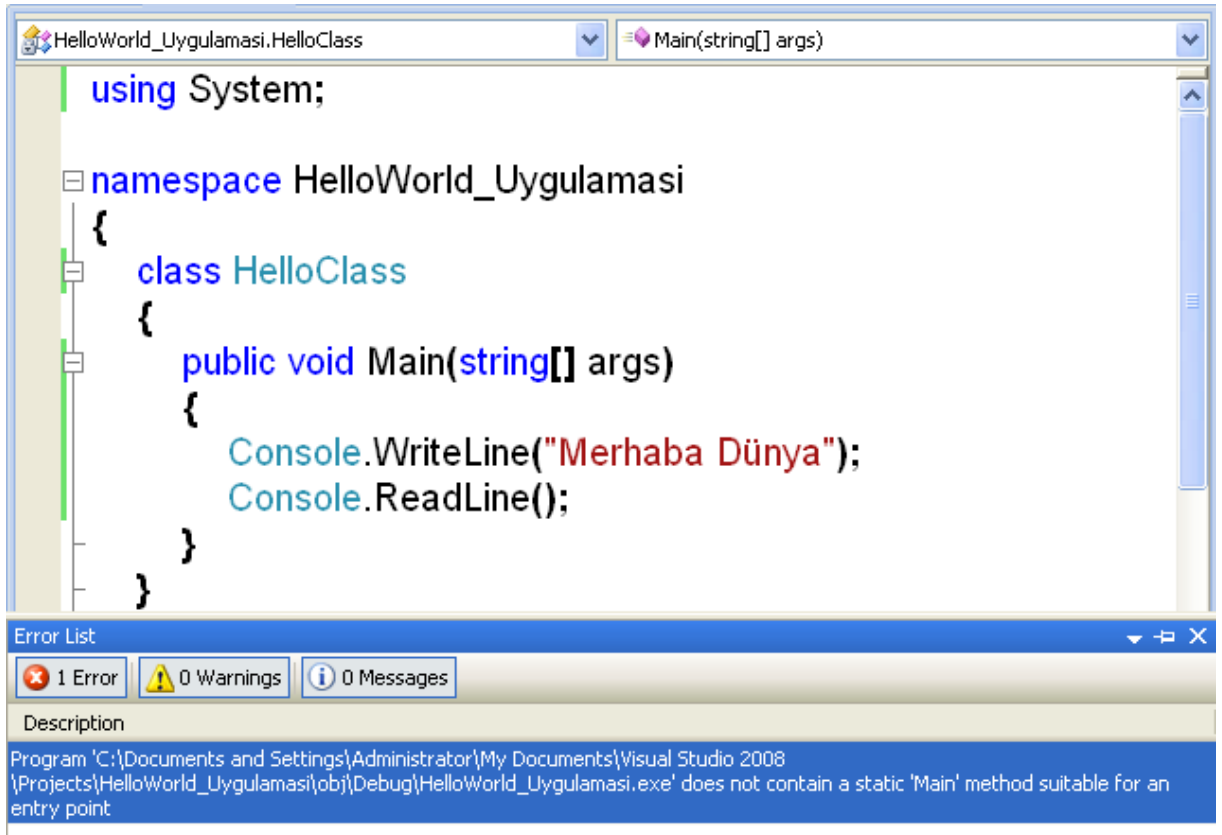
Her uygulamanın bir başlangıç noktası olmalıdır. Bir C# uygulaması çalıştırıldığı zaman çalışmaya **Main** adındaki metottan başlar, kontrol bu metodun sonuna geldiğinde ise uygulama sonlanır. (Ya da varsa metot içerisinde **return** ifadesinin görüldüğü yerde uygulama sonlanır)

C#, büyük-küçük harf duyarlı (case-sensitive) bir dil olduğu için "Main" ile "main", (Benzer bir örnek vermek gerekirse "ReadLine" ile "Readline") aynı değildir. Dolayısıyla başlangıç metodu her zaman ilk harfi büyük, geri kalan harfleri küçük "**Main**" olacak şekilde kodlanmalıdır.

Bir C# uygulamasında birden fazla sınıf olabilir, ancak sadece bir tane giriş noktası olmalıdır. Aynı uygulamada her birinde Main olan birden çok sınıf da yer alabilir; ancak sadece bir tane Main çalıştırılacaktır. O yüzden uygulama derlendiğinde hangisinin kullanılacağı belirtilmelidir.

Main'in niteleyicileri de önemlidir. Koda bakılacak olursa, Main metodunun tanımlamasındaki niteleyiciler "public", "static" ve "void" olarak belirlenmiştir. Bu anahtar kelimeler sonraki modüllerde daha ayrıntılı olarak incelenecek; o zamana kadar "public" üyelerin diğer tipler ve üyeler tarafından erişilebilir, static üyelerin sınıf düzeyinde üyeler olduğunu ve "sınıfAdı.üyeAdı" şeklinde çağrılarak kullanılabileceğini, "void" metotların ise çalışması sona erdiğinde ortama bir değer döndürmediğini bilmek yeterlidir.

Main() metodunun erişim belirleyicisi "public" yerine "private" da olabilir; bu şekilde diğer assembly'lerin, yazdığımız uygulamanın giriş noktasını çağırmasını sağlamış olur. (Visual Studio, bir programın Main() metodunu otomatik olarak "private" tanımlar) Ayrıca dönüş tipi de "void" yerine "int" olabilir. Bu, uygulama geliştiriciye Main() metodunun başarıyla sonlanıp sonlanmadığını öğrenmek için geriye sayısal değer döndürme fırsatı sağlar. Ancak Main() metodu her zaman "**static**" olmak zorundadır; yoksa derleyici tarafından uygulama için uygun bir giriş noktası bulunamaz.



**Main() metodu static olmalıdır.**



**EĞİTİM :**

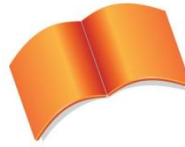
**MERHABA DÜNYA**

**Bölüm :**

**Using Direktifi ve System  
İsim Alanı**

**Konu :**

Namespace (İsim Alanı  
Kavramı)

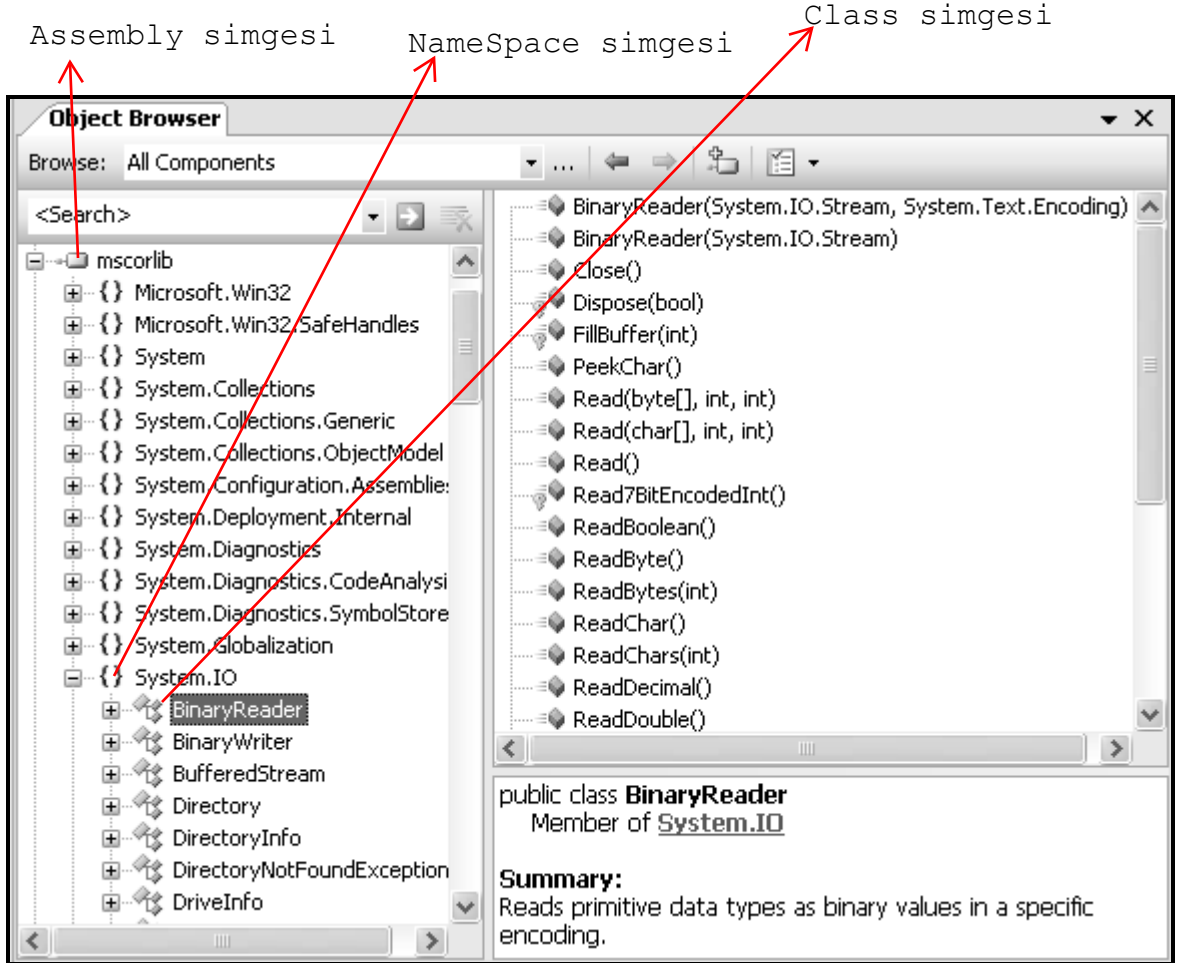


Microsoft Türkiye

**Açık Akademi**

## Namespace (isim alanı) Kavramı

C#, Microsoft.NET Framework'ün bir parçası olarak temel ve faydalı programlama işlerini gerçekleştirmek için yazılım geliştiricinin hizmetine birçok kullanışlı sınıf sunmuştur. Bu sınıflar isim alanları (namespaces) altında organize edilmişlerdir. Bir namespace, benzer amaca hizmet eden sınıflardan oluşan bir kümedir. Aynı zamanda bir isim alanı, başka isim alanlarını da içerebilir. Mesela System.IO isim alanı dosya giriş/çıkış ile ilgili tipleri, System.Data isim alanı temel veritabanı işlemleri ile ilgili tipleri içerir. Ayrıca vurgulamak da fayda var ki; tek bir assembly (mscorlib.dll gibi) istenilen sayıda isim alanı içerebilir, her bir isim alanında da istenilen sayıda tip yer alabilir. Bu, Visual Studio 2005 içerisindeki View menüsünden erişilebilecek olan Object Browser yardımıyla görülebilir:



Tek bir assembly, birden fazla isim alanına sahip olabilir.

**EĞİTİM :**

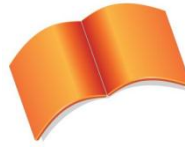
**MERHABA DÜNYA**

**Bölüm :**

**Temel Giriş-Çıkış İşlemleri**

**Konu :**

**Console Sınıfı**



Microsoft Türkiye

**Açık Akademi**

## Console Sınıfı

Kitabın şu ana kadarki örneklerinde **System.Console** sınıfı sık sık kullanıldı, kullanılmaya da devam edilecek. Konsol kullanıcı arayüzü, Windows kullanıcı arayüzü kadar çekici değilken örnekleri konsol arayüzü ile kısıtlamak bize grafik arayüzünün kompleksliği ile uğraşmak yerine öğrenmeye çalıştığımız C# temellerine odaklanma fırsatı sunuyor.

Adından anlaşılabilceği gibi bu sınıf konsol uygulamalarında kullanıcıdan veri alma, kullanıcıya veri gösterme, oluşan hata durumlarında gerekli manipülasyonları gerçekleştirmeyi üstlenir. Aşağıdaki üyeler örnek olarak verilebilir...

Üye	Tanımı
<b>BackgroundColor</b> <b>ForegroundColor</b>	Ekran çıktısının arka plan ve yazı rengini değiştirmeye yarar.
<b>WindowHeight</b> <b>WindowWidth</b> <b>WindowTop</b> <b>WindowLeft</b>	Konsol ekran boyutunu değiştirmeye yarar.
<b>BufferHeight</b> <b>BufferWidth</b>	Konsol ekranında yazı yazılabilecek alanı belirler.
<b>Clear()</b>	Konsoldaki yazı alanını temizleyen metot.
<b>Title</b>	Çalışan konsol penceresinin başlığını belirler.

**EĞİTİM :**

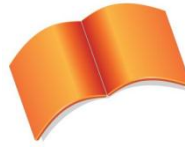
**MERHABA DÜNYA**

**Bölüm :**

**Temel Giriş-Çıkış İşlemleri**

**Konu :**

**WriteLine - ReadLine**



Microsoft Türkiye

**Açıık Akademi**

Tablodakilere ek olarak **Console** sınıfı, ekrandan yapılan giriş ve çıkışları yakalayan üyeler tanımlar. Bu üyelere sınıf adı üzerinden erişiriz ve daha sonra görüleceği üzere bu üyelere static üyeler denir. **WriteLine()**, ekrana metin olarak yazı yazılmasını sağlar ve imleci bir alt satırda beklemeye zorlar, böylece arkasından yazılacak metin bir alt satırdan başlar. **Write()** metodunun tek farkı ise imleci ekrana yazılan yazının hemen sonunda bekletmesidir, böylece arkasından yazılacak metin aynı satırdan devam edecektir. **ReadLine()**, ekrana yazılan yazıyı metin olarak satır sonuna kadar okumayı sağlar, **Read()** ise ekrana yazılan metnin sadece ilk karakterini okur ve bunu karakterin ASCII karakter setindeki sayısal karşılığı olarak döndürür (Daha sonra görülecek dönüştürme yöntemleri ile bunun karakter karşılığı elde edilebilir), bir döngü içerisinde okunduğunda ve okuyacak karakter kalmadığında ise -1 döndürür. ReadLine() ve Read() metodu, ekrandaki yazıları kullanıcı klavyesinden ENTER tuşuna basılınca okur; yani uygulama, kaynak kodunda bu iki metoda rastlarsa ENTER tuşlanana kadar bekleyecektir.

**EĞİTİM :**

**MERHABA DÜNYA**

**Bölüm :**

**Temel Giriş-Çıkış İşlemleri**

**Konu :**

**Console Sınıfı ile Main()**

**Metodu**

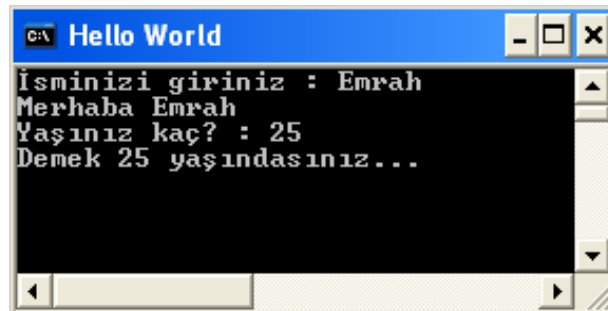


Microsoft Türkiye

**Açık Akademi**

*Console* sınıfı ile basit I/O işlemlerini örneklemek için kullanıcıdan veri alıp bu verileri yeniden ekrana yazan aşağıdaki gibi bir `Main()` metodu yazılabilir:

```
...  
  
public static void Main(string[] args)  
{  
    //Ekrana yazı yazıyoruz...  
    Console.Write("İsminizi giriniz : ");  
    //Kullanıcı ENTER'a basınca o ana kadar yazılan yazı okunur ve bir değişkene atılır.  
    string giris = Console.ReadLine();  
    //Değişken üzerindeki bilgiyi ekrana yazıyoruz...  
    Console.WriteLine("Merhaba {0}",giris);  
  
    Console.Write("Yaşınız kaç? : ");  
    giris = Console.ReadLine();  
    Console.WriteLine("Demek {0} yaşındasınız...",giris);  
}  
  
...
```



Basit I/O işlemleri



**EĞİTİM :**

**MERHABA DÜNYA**

**Bölüm :**

**Temel Giriş-Çıkış İşlemleri**

**Konu :**

**Yer Tutucu Place Holder**



Microsoft Türkiye

**Açık Akademi**

Şu ana kadarki örneklerde WriteLine() içerisine yazılan birçok {0}, {1} kullanımlarını, text alanının ardından da değerlerinin verildiği görüldü. .NET, string formatlamanın yeni bir stilini geliştirdi, buna “yer tutucu” (Placeholder) denir.

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

    int sayisal = 5;
    string text = "Merhaba";
    bool mantiksal = true;

    // Bir textin içerisinde yazılan '\n' yeni bir satır ekler.
    Console.WriteLine("Sayısal değer : {0}\nText değer : {1}\n
    Mantıksal değer : {2}", sayisal, text, mantiksal);

    Console.ReadLine();
}
```



String ifade içerisinde birden çok “yer tutucu” kullanımı

WriteLine() içerisine çift tırnak ("...") arasında yazılan her şey normalde ekranda görünür. Bunun istisnai durumlarından biri kullanılan süslü parantezlerdir. Bu süslü parantezler ekran çıktısında görünmez; çünkü onlar başka değerler için yer tutarlar. Yer tutmak istediğimiz her değer için bir süslü parantez içerisine '0'dan başlamak kaydıyla indeks numaraları verilir. Bu indekslerin yerine ekranda görünecek değerler ise çift tırnakların arkasından verilir ("...",değerler). Eğer birden fazla yer tutucu kullanılmışsa değerler, virgülle ayrılarak içerdeki indeks sırasına göre yazılır.

**EĞİTİM :**

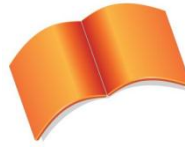
**MERHABA DÜNYA**

**Bölüm :**

**Temel Giriş-Çıkış İşlemleri**

**Konu :**

**Yer Tutucu'nun Birden Fazla  
Yerde Kullanılması**



Microsoft Türkiye

**Açık Akademi**

Aynı zamanda tek bir yer tutucunun, tek bir string veri ile birden fazla yerde kullanılması da mümkündür. Örneğin “asla, asla vazgeçemem senden asla” şarkı sözlerini koda dökmek istersek;

```
//Tarkan söylüyor...
```

```
Console.WriteLine("{0}, {0} vazgeçemem senden {0}", "asla");
```

Yer tutucu indeksine eklenecek bir parametre ile alan genişliği belirlenebilir ve ilgili değerin bu alanda sağa ya da sola dayalı olarak yazdırılması sağlanabilir.

```
Console.WriteLine("\t10 birimlik alanda sola dayalı :{0,-10}\t",23);
```

```
Console.WriteLine("\t10 birimlik alanda sağa dayalı :{0,10}\t",23);
```

Bir string verinin içerisinde kullanılan ‘\’i şareti, kendisinden sonra gelen karakterin özel anlamını iptal edip metin olarak ekrana yazdırır. Mesela ‘\’, ekrana ‘\’ yazılmasını, ‘\’ ekrana ‘\’ yazılmasını sağlar. Buna alternatif olarak string verinin arasına yazıldığı çift tırnağın önüne ‘@’ işareti koyularak verinin kelimesi kelimesine ekrana yazılması sağlanabilir. Örneğin @“c:\\Documents and Settings\\Administrator” olarak kodlamak, ekrandaki çıktının “c:\\Documents and Settings\\Administrator” olmasını sağlar.

**EĞİTİM :**

**MERHABA DÜNYA**

**Bölüm :**

**Temel Giriş-Çıkış İşlemleri**

**Konu :**

**Nümerik Formatlama**



Microsoft Türkiye

**Açık Akademi**

## Nümerik Formatlama

Nümerik verilerin nasıl formatlanıp görüntüleneceğini belirtmek için “format stringi” kullanılır. Tam söz dizimi (syntax) **{N,M,:FormatString}** iken burada N yer tutucunun indeks numaralandırıcısını, M alan genişliği ve önüne koyulan ‘+’ – ‘-’ işaretiyle yazının hangi tarafa dayalı yazılacağını, FormatString ise nümerik verinin nasıl gösterilmesi gerektiğini belirtir. Bütün format seçenekleri aşağıdadır:

Format Karakteri	Açıklaması
<b>C ya da c</b>	<b>Nümerik</b> veriyi lokal para sembolünü kullanarak para olarak gösterir. Format string harfini “Currency” kelimesinin baş harfinden alır.
<b>D ya da d</b>	<b>Bu</b> format sadece tam sayılar için desteklenir. Hemen arkasına aldığı sayı ile (d5 gibi) ekranda gösterilmesi gereken minimum basamak sayısı belirtilebilir ve eğer sayının basamak sayısı, belirtilen minimum basamak sayısından fazlaysa sayının başına ‘0’ eklenerek stringe çevrilir. Format string harfini “Decimal” kelimesinin baş harfinden alır.
<b>E ya da e</b>	<b>Sayıyı</b> üstel notasyon kullanarak göstermek için kullanılır. Sayıyı “d.dddd...E+ddd” formatında stringe çevirir. (d’ ler birer rakamı temsil ediyor) Burada ilk basamak her zaman tam sayı olur ve format stringinin hemen arkasına belirtilebilecek bir sayı ile ondalık kısmında kaç basamak bulunacağı belirtilebilir. Belirtilmezse varsayılan olarak bu değer 6’dır. Üstel ifade ise her zaman minimum 3 basamaktan oluşur; dolayısıyla eğer ihtiyaç duyulursa e ya da E (string format’ına yazılacak harfin büyük mü küçük mü olduğuna bağlı olarak değişiyor)’den sonraki rakam “0” larla 3 basamağa tamamlanır... Format string harfini “Exponential” kelimesinin baş harfinden alır.
<b>F ya da f</b>	<b>Sayı</b> , ondalık olsun ya da olmasın varsayılan olarak 2 basamak ondalıkla stringe çevrilir. Eğer tam sayı ise sonuna ondalık olarak sıfırlar eklenir. Eğer ondalık sayı ise ve ondalık kısmı ikiden fazla ise varsayılan değer iki olduğu için virgülden sonraki değerlerinin sadece ilk ikisi alınarak formatlama yapılır... Format string’inin hemen arkasından verilecek bir değer ile sayının ondalık kısmında bulunması istenilen basamak sayısı belirlenebilir (f4 gibi). Format string harfini “Fixed” kelimesinin baş harfinden alır.
<b>G ya da g</b>	<b>Sayıyı</b> hemen arkasından verilebilen değere ve basamak sayısına bağlı olarak ya aynen yazılır ya da ‘e’ string formatıyla stringe çevrilir. Eğer format string ile birlikte bir değer verilmezse sayı olduğu gibi alınır. Eğer verilirse; değer, stringe çevrilecek sayının istenilen tam ve ondalık basamak sayıları toplamını işaret eder. Değer, sayının tam kısmındaki basamak sayısından az ise formatlama üstel olarak yapılır, diğer durumlarda sayı olduğu gibi çevrilir. Format string harfini “General” kelimesinin baş harfinden alır.

<b>N ya da n</b>	<b>Bu</b> format stringi ile soldan başlayarak her 3 basamakta bir araya bin ayırıcı koyulması sağlanır. Hemen arkasından verilebilecek değer ile istenilen ondalık basamak sayısı verilebilir. Bu değer varsayılan olarak 2'dir. Format string harfini "Number" kelimesinin baş harfinden alır.
<b>P ya da p</b>	<b>Sayı</b> 100 ile çarpılarak yüzde işareti ile sunulur. Hemen arkasından verilebilecek değer, yüzde değerinden sonra gelmesi istenilen ondalık basamak sayısıdır. Bu değer varsayılan olarak 2'dir. Format string harfini "Percent" kelimesinin baş harfinden alır.
<b>X ya da x</b>	<b>Sayı</b> hexadecimal (onaltılı sayı sistemi) formatta stringe çevrilir. Sadece tam sayılar için desteklenir. Format string harfini "Hexadecimal" kelimesindeki x harfinden alır.

---

#### .NET formatlama stringleri ve anlamları

Aşağıda nümerik formatlama ile ilgili bazı örneklere yer verilmiştir:

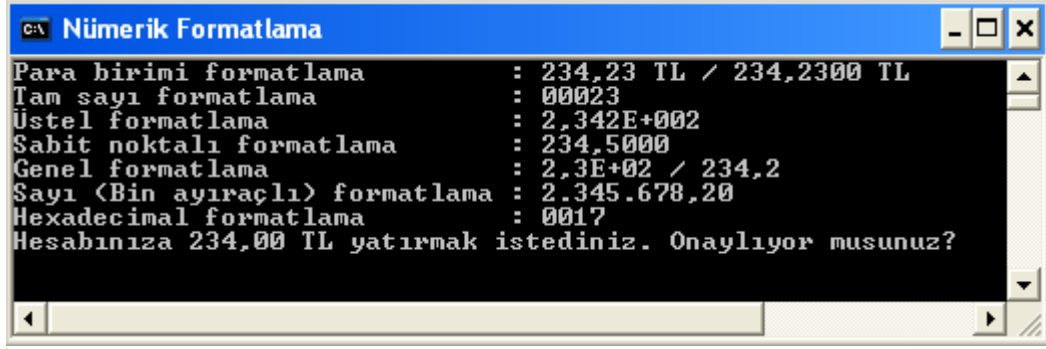
```
public static void Main(string[] args)
{
    Console.Title = "Nümerik Formatlama";
    Console.WriteLine("Para birimi formatlama: {0:C} / {0:C4}", 234.23);
    Console.WriteLine("Tam sayı formatlama : {0:D5}",23);
    Console.WriteLine("Üstel formatlama : {0:E3}",234.2);
    Console.WriteLine("Sabit noktalı formatlama : {0:f4}",234.5);
    Console.WriteLine("Genel formatlama : {0:G2} {0:G4}",234.23);
    Console.WriteLine("Sayı(Bin ayraç) formatlama : {0:N}",2345678.2);
    Console.WriteLine("Hexadecimal formatlama : {0:X4}",23);
    Console.ReadLine();
}
```

.NET formatlama karakterlerinin kullanımı sadece konsol uygulamaları ile sınırlı değil. **String.Format()** metodu ile bütün format stringleri herhangi bir uygulama tipinde (Masaüstü uygulamaları, ASP.NET, XML Web servisleri...) kullanılabilir.

```
public static void Main(string[] args)
{
    ...
    //Yeni bir string veri oluşturmak için string.Format() metodu kullanılıyor.
    string formatStr;
    formatStr = string.Format("Hesabınıza {0:C} yatırmak istediniz. Onaylıyor musunuz?",234);
    Console.WriteLine(formatStr);

    Console.ReadLine();
}
```

Bu kodların çalıştırılmasıyla elde edilen ekran çıktısı ise aşağıdadır:



Nümerik formatlama örnekleri



**EĞİTİM :**

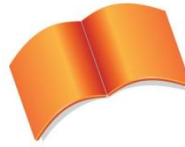
**MERHABA DÜNYA**

**Bölüm :**

**Temel Giriş-Çıkış İşlemleri**

**Konu :**

**Uygulamalarda Yorum Satırı**



Microsoft Türkiye

**Açık Akademi**

## Uygulamalarda Yorum Satırı

Uygulamalar için dokümantasyon sağlamak önemlidir. Yazılan kod ile ilgili yeterli bilgi sağlamak, geliştirilme sürecinde hiç yer almamış veya sonradan katılmış bir programcının uygulamayı anlayıp takip edebilir seviyeye gelmesi sürecini fark edilir düzeyde etkiler. İyi yorumlar, sadece kodlara bakılarak anlaşılması kolay olmayacak bilgiyi sağlarlar ve orada verilmeye çalışılan cevap “NE” değil “NEDEN” sorusu olmalıdır.

C#, uygulama kodlarına yorum eklemek için şu yolları sunar: Tek yorum satırı, çoklu yorum satırları ve XML dokümantasyonu. Tek satırlık yorumlar çift bölü işareti (//) ile eklenebilir. Uygulama çalıştırıldığı zaman bu karakterlerin arkasından gelen yazılar satır sonuna kadar ihmal edilir.

```
//Uyelik için kullanıcıdan ismi alınır  
Console.Write("Adınızı giriniz : ");  
string ad = Console.ReadLine();
```

Eğer aynı zamanda birden fazla satır yorumlanmak istenirse ya her satıra çift bölü işareti (//) koyulur ya da blok yorumlama işaretinden faydalanılır. Blok yorum, ‘/\*’ işareti ile başlayıp ‘\*/’ işaretini görene kadar devam eder.

```
/*Aşağıdaki işlemin sonucu  
x değişkenine atanıp kullanılır*/  
int x = (...);
```

**EĞİTİM :**

**MERHABA DÜNYA**

**Bölüm :**

**C# Derleyicisi Kullanarak  
Uygulama Derleme**

**Konu :**

**C# Komut Satırı Derleyici İle  
Çalışmak**



Microsoft Türkiye

**Açık Akademi**

## C# Komut Satırı Derleyicisi ile Çalışmak

C# uygulaması çalıştırılmadan önce **derlenmelidir**. Bu iş için Visual Studio gibi uygulama geliştirme ortamı kullanılabileceği gibi C# uygulamaları için **csc.exe** ile de .NET assembly'leri oluşturabilir (csc, C-Sharp Compiler'ı simgeliyor). Bu derleme aracı .NET Framework 4.0 SDK'nın (Software Development Kit –Yazılım Geliştirme Kiti-) içerisinde yer almaktadır. Büyük ölçekli bir uygulama hiçbir zaman komut satırından konsol penceresi kullanarak derlenmez; ancak \*.cs uzantılı dosyanın nasıl derleneceğinin temellerini bilmek bir uygulama geliştirici açısından önemlidir. Ayrıca böyle bir sürece ihtiyaç duyulması için birkaç sebep göstermek gerekirse;

- Uygulamanın bulunduğu ortamda Visual Studio bulunmayabilir.
- C# bilgimizi biraz daha derinleştirmek isteyebiliriz. Böylece otomatik derleme aracı yazma hakkında temel bilgimiz artmış olur.
- Bir uygulamayı derlemek için grafik bir arabirim kullanılsa da eninde sonunda csc.exe'ye başvurulacaktır. Dolayısıyla burada arka planda neler olup bittiği ile ilgili biraz daha ayrıntılı bilgi elde etmek istenebilir. (Örneğin kendi Visual Studio IDE'mizi yazmak istediğimizde...) ! ! ! . . .

**EĞİTİM :**

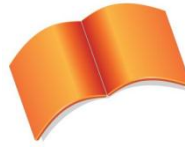
**MERHABA DÜNYA**

**Bölüm :**

**C# Derleyicisi Kullanarak  
Uygulama Derleme**

**Konu :**

**Csc.exe'nin Tanınması**



Microsoft Türkiye

**Açık Akademi**

C# komut-satırı derleyicisi kullanılmaya başlanmadan önce çalışılan makinenin csc.exe'yi tanıdığından emin olunması gerekiyor. Eğer makine doğru bir şekilde konfigüre edilmediyse C# dosyalarını derlemek için her seferinde csc.exe'nin bulunduğu dizini belirtmek gerekecek. Makinemize \*.cs uzantılı dosyalarımızı her hangi bir dizinden derleme yeteneği kazandırmak için (Windows XP işletim sistemi üzerinde gerçekleştirilen) aşağıdaki adımları takip etmek gerekir:

- Bilgisayarıma (MyComputer) sağ tıklanır ve Seçenekler (Properties) penceresi açılır.
- Gelişmiş (Advanced) tabı açılır ve oradan Ortam değişkenleri (Environment Variable) butonu tıklanır.

“Sistem Değişkenleri” bölümünden “Path” değişkenine çift tıklanır ve listenin en sonuna csc.exe'nin içerisinde bulunduğu klasörün fiziki yolu yazılır. (“Path” değişkenindeki herbir değer noktalı virgül ile ayrılmıştır)

**EĞİTİM :**

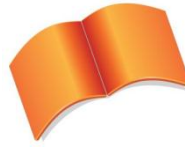
**MERHABA DÜNYA**

**Bölüm :**

**C# Derleyicisi Kullanarak  
Uygulama Derleme**

**Konu :**

**Csc.exe'yi Çalıştırmanın  
Adımları**



Microsoft Türkiye

**Açık Akademi**

Uygulama geliştirme bilgisayarı csc.exe'yi tanıdığına göre C# komut satırı derleyicisi ve notepad kullanarak DemoUyg.exe adında basit bir tek dosyalı assembly geliştirilebilir. Bir notepad açarak aşağıdakiler yazılır:

```
//Basit bir C# uygulaması
using System;

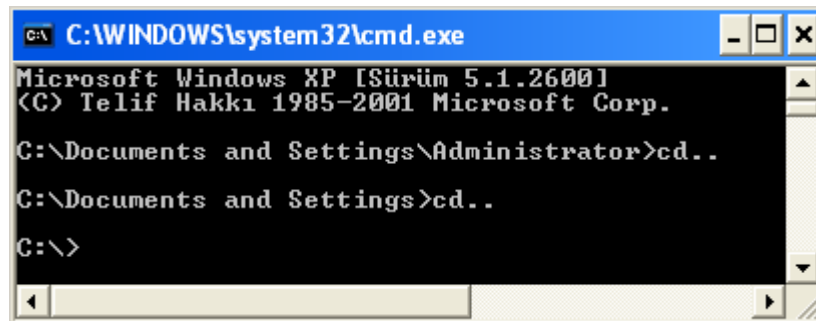
class DemoUyg
{
    public static void Main(string[] args)
    {
        Console.WriteLine("Deneme 1,2");
        Console.ReadLine();
    }
}
```

Dosya uygun bir dizine (Mesela c:\Ornekler klasörüne) DemoUyg.cs olarak kaydededilir. C# derleyicisinin seçenekleri incelenecek olursa dikkat edilecek ilk nokta; oluşturulacak assembly'nin adı ve tipinin nasıl belirtileceğidir (Örneğin Uygulamam.exe adında bir konsol uygulaması, Robotik.dll adında bir kod kütüphanesi, WinUygulamam.exe adında bir masaüstü uygulaması vb...) Her olasılık, csc.exe'ye komut satırı parametresi olarak geçirilen özel bir işaret ile temsil edilir.

Seçenek	Tanımı
<b>/out</b>	Oluşturulan assembly'ye isim vermek için kullanılır. Varsayılan olarak oluşan assembly'nin adı, başlangıç input'u olan *.cs dosya adıdır.
<b>/target:exe</b>	Çalıştırılabilir konsol uygulaması geliştirmek için bu seçenek kullanılabilir. Varsayılan çıktı tipi budur.
<b>/target:library</b>	Bu seçenek tek dosyalı bir *.dll assembly'si inşa edilir.
<b>/target:module</b>	Bu seçenek bir <b>module</b> oluşturur. Her bir modül, çok dosyalı bir assembly'nin elemanıdır.
<b>/target:winexe</b>	Target:exe ile masaüstü uygulamaları geliştirilebilirken, bu seçenek arka planda konsol penceresinin kapatılması gerektiğini garanti eder.

**Tablo 4: C# derleyicisinin çıktı tabanlı seçenekleri**

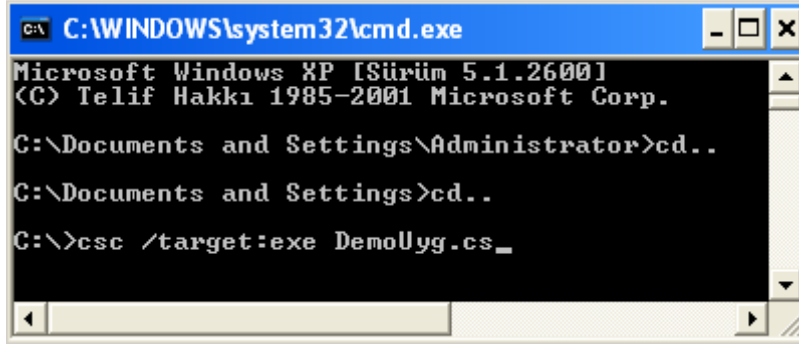
DemoUyg.cs'i DemoUyg.exe ismindeki çalıştırılabilir bir uygulamaya derlemek için komut penceresinden \*.cs dosyanın dizinine geçilir:



Derlenecek dosyanın dizinine inilir.

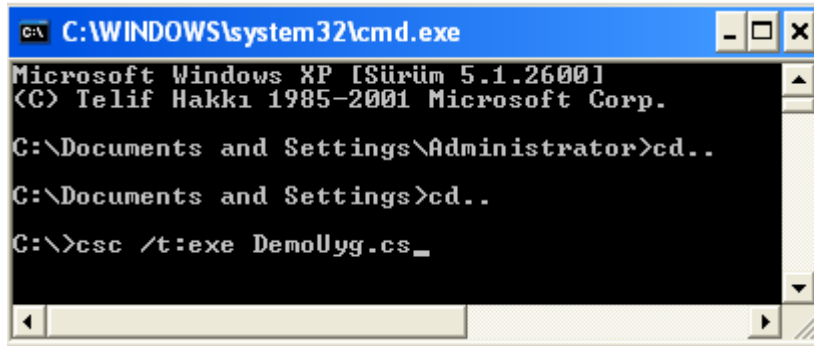
ve aşağıdaki komut satırı girilip ENTER'a basılır:





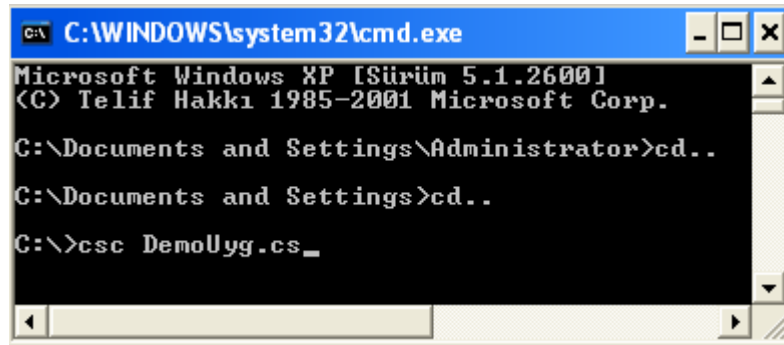
**DemoUyg.cs, aynı isimli çalıştırılabilir (.exe) dosyaya derleniyor.**

Burada bilinçli olarak /out kullanılmadı. O yüzden programın giriş noktasını içeren (Main()) metodu DemoUyg sınıfından üretilen çalıştırılabilir dosya \*.cs dosyasıyla aynı isimde (DemoUyg.exe olarak) ilgili dizinde oluşturulur. Bu arada birçok C# işareti, kısaltmaları tanır. Burada da /target yerine /t kullanılabilir:



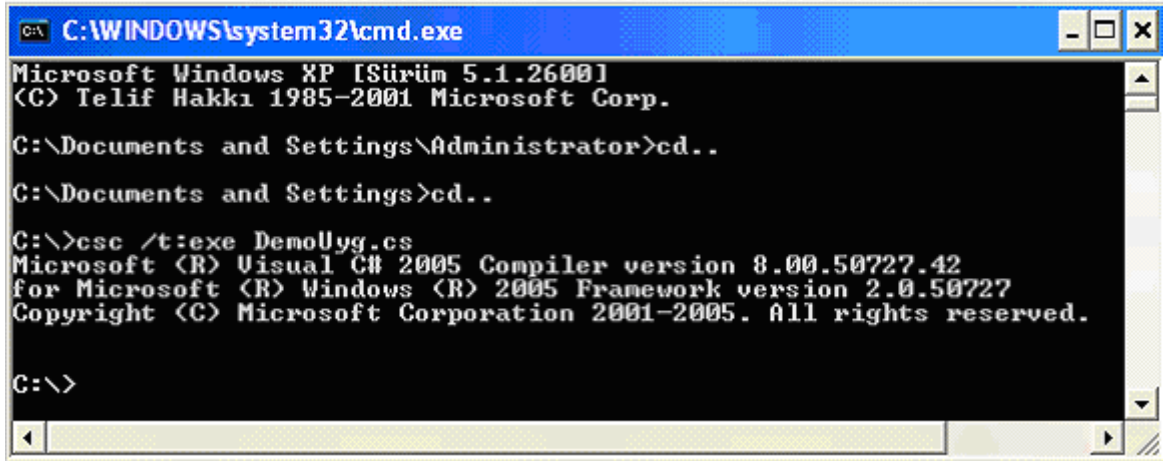
**Derleme /target:exe yerine kısaca /t:exe şeklinde de yapılabilir.**

Aynı zamanda bu işaret (t:exe) C# derleyicisinin varsayılan dosya çıktı tipidir; dolayısıyla DemoUyg.cs şu şekilde de derlenebilir.



**Varsayılan /target:exe olduğu için hiçbir dosya çıktı tipi belirtilmediğinde target:exe olarak işlem yapılır.**

Artık üretilen DemoUyg.exe komut satırından ya da üretildiği dizinden çalıştırılabilir. Ekran çıktıları aşağıdaki gibidir:



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Sürüm 5.1.2600]
(C) Telif Hakkı 1985-2001 Microsoft Corp.

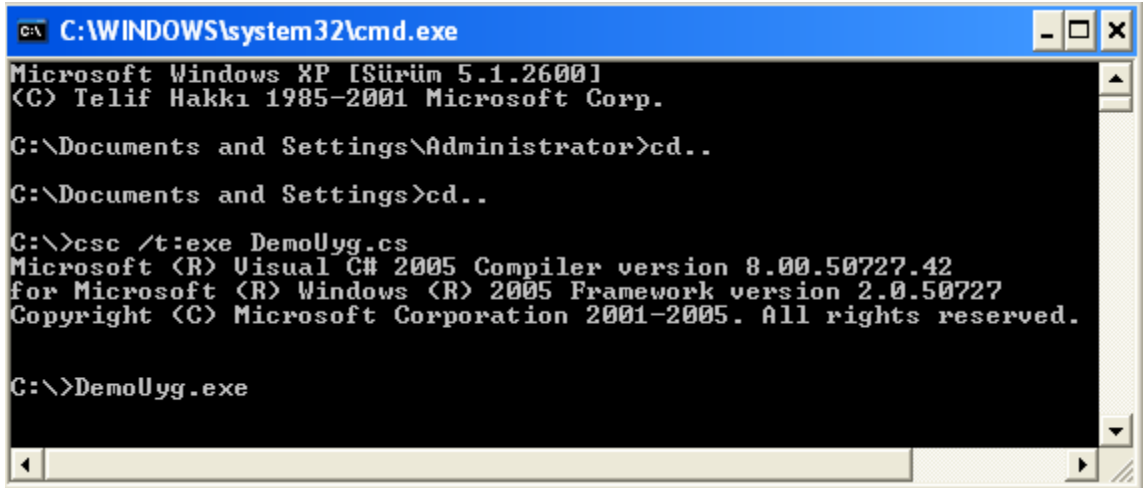
C:\Documents and Settings\Administrator>cd..
C:\Documents and Settings>cd..

C:\>csc /t:exe DemoUyg.cs
Microsoft (R) Visual C# 2005 Compiler version 8.00.50727.42
for Microsoft (R) Windows (R) 2005 Framework version 2.0.50727
Copyright (C) Microsoft Corporation 2001-2005. All rights reserved.

C:\>
```

Sayfa başarılı bir şekilde derlenip programın çalıştırılabilir assembly'si (DemoUyg.exe) oluşturulur...

İster 'c' dizini altındaki dosyaya gidip, ister komut-satırından .exe dosyasının adı yazılıp ENTER'a basılarak çalıştırılabilir.



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Sürüm 5.1.2600]
(C) Telif Hakkı 1985-2001 Microsoft Corp.

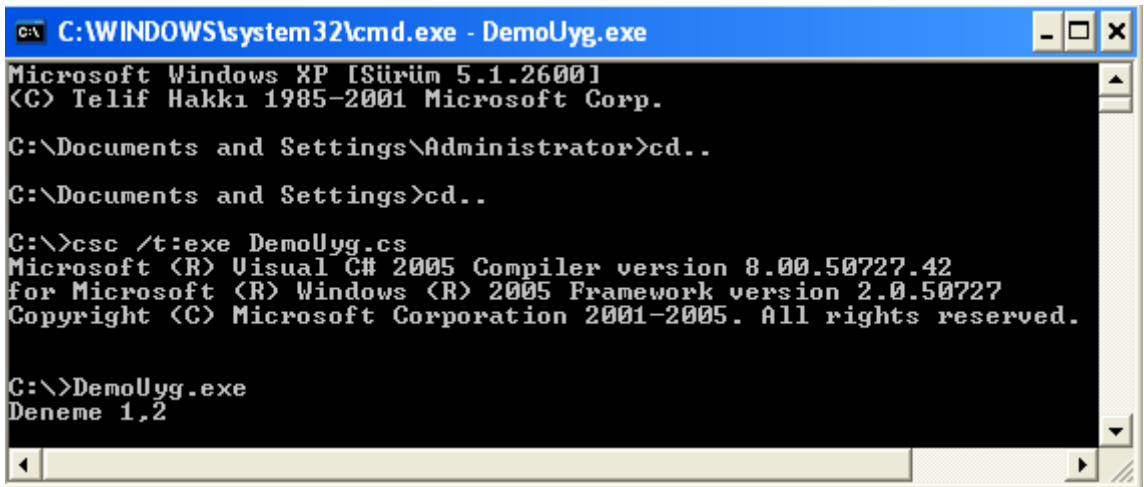
C:\Documents and Settings\Administrator>cd..
C:\Documents and Settings>cd..

C:\>csc /t:exe DemoUyg.cs
Microsoft (R) Visual C# 2005 Compiler version 8.00.50727.42
for Microsoft (R) Windows (R) 2005 Framework version 2.0.50727
Copyright (C) Microsoft Corporation 2001-2005. All rights reserved.

C:\>DemoUyg.exe
```

Derleme sonucunda oluşan .exe dosyasının çalıştırılması

Ve son çıktı aşağıdaki gibi olur:



```
C:\WINDOWS\system32\cmd.exe - DemoUyg.exe
Microsoft Windows XP [Sürüm 5.1.2600]
(C) Telif Hakkı 1985-2001 Microsoft Corp.

C:\Documents and Settings\Administrator>cd..
C:\Documents and Settings>cd..

C:\>csc /t:exe DemoUyg.cs
Microsoft (R) Visual C# 2005 Compiler version 8.00.50727.42
for Microsoft (R) Windows (R) 2005 Framework version 2.0.50727
Copyright (C) Microsoft Corporation 2001-2005. All rights reserved.

C:\>DemoUyg.exe
Deneme 1,2
```

Çalışan programın ekran çıktısı

\*.cs dosyasında bir deęişiklik yapılmadıęı sürece uygulama, üretilen bu \*.exe dosyasından çalıştırılabilir. \*.cs dosyasında bir deęişiklik yapıldığında ise uygulama, güncel hali ile çalışmaya devam etsin diye yeniden derlenmelidir.

**EĞİTİM :**

**MERHABA DÜNYA**

**Bölüm :**

**C# Derleyicisi Kullanarak  
Uygulama Derleme**

**Konu :**

**Framework Versiyonu ve Path  
Değişkeni Güncellenmesi**



Microsoft Türkiye

**Açık Akademi**

---

C:\WINDOWS\Microsoft.NET\Framework\v4.0.30319

---

Tabiki sizlerin adresi kullandığınız Framework'ün versiyonu ve .NET Framework 4.0 SDK'sının dizininizdeki yerine bağlı olarak değişebilir. "Path" değişkeni güncellendikten sonra test etmek amacıyla açık olan bütün konsol pencereleri kapatılıp yeni bir tane açılır ve komut penceresindeyken

---

csc ?/ ya da csc -?

---

yazılır. Eğer işler yolunda gitmişse C# derleyicisinin sunduğu seçenekler listesinin elde edilmiş olması gerekmektedir. Bu değişkenin eklenmesiyle artık herhangi bir .NET aracını herhangi bir komut penceresinden çalıştırmak mümkündür.