1. NESNE TABANLI PROGRAMLAMA

**1.1. Nesne Tabanlı Programlama Nedir?**

Nesne tabanlı programlamada esas olan, gerçek hayatta var olan olguların programlamaya aktarılmasındaki yeni yaklaşımdır. Klasik prosedürel programlamada verilerimiz ve fonksiyonlarımız vardı. Yani her şey veri ve bu veriyi işleyen metotlar etrafında dönüyordu.

Aslında nesne tabanlı programlamada da iki önemli birim; veri ve veriyi işleyip mantıklı sonuçlar üreten metotlar bulunur. Ama buradaki fark gerçek hayattaki olguların da daha iyi gözlenip programlama dünyasına aktarılmasındadır.

Eğer ütü ile ilgili bir program yapmış olsak ve nesne tabanlı programlama tekniğini kullansak hemen bir ütü sınıfı (class) oluştururduk. Bu sınıfta ütüye ait bilgiler (veriler) ve ütü ile yapabileceğimiz işler (metot) bulunurdu. O zaman nesne tabanlı programlamada bir sınıfta, sınıfa ait veriler ve bu verileri işleyip bir takım faydalı sonuçlar üreten fonksiyonlar /metotlar bulunur.

Nesne tabanlı programlamanın bazı temel kavramları vardır:

* + 1. **Kapsülleme (Encapsulation)**

İşlemlerin ve verilerin gizlenmesine veya bir grup içerisinde toplanmasına kapsülleme denir. Kapsülleme sayesinde, sınıfımız içinde kullandığımız alanların dışarıdan herhangi bir etki ile doğrudan değiştirilmelerini engellemiş oluyoruz.

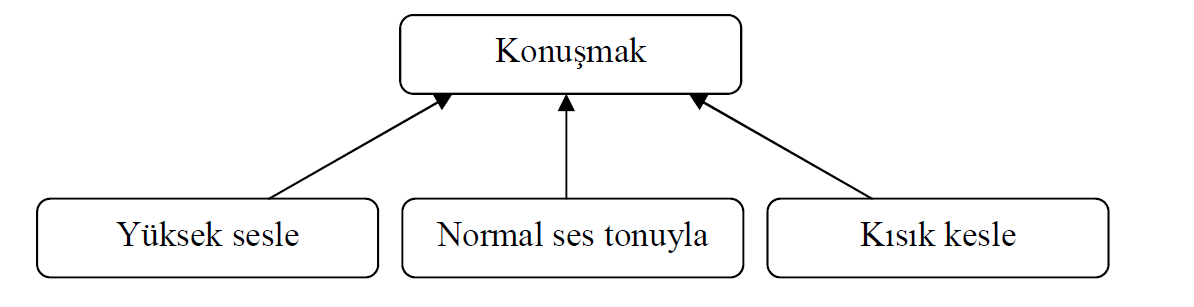
Örneğin: saat içinde çok karmaşık mekanizmalar bulunmaktadır. Ama bu karmaşık mekanizma saatin kullanıcısını ilgilendiren bir olay değildir. Kullanıcı için önemli olan saatin düzgün çalışmasıdır.

**1.1.2. Kalıtım (Inheritance)**

En genel tanımı ile kalıtım, **"Bir sınıftan yeni sınıflar türetmektir"** Herşeyden önce kalıtım yolu ile bir sınıftan yeni sınıflar türetmenin, türetilen sınıflara etkisi nedir? Bu sorunun cevabı kalıtımın da özünü oluşturmaktadır. **Türetilen her bir sınıf, türediği sınıfın özelliklerini de devralır**. Buradan, türetilmiş bir sınıf içerisinden, türediği sınıfa ait üyelere erişilebileceği sonucunu çıkartabiliriz. Elbette bu erişimin de bazı kuralları vardır. Örneğin erişim belirleyicilerinin etkisi veya aynı üyelerin kullanılışı gibi durumlar.



**1.1.3. Çok Biçimlilik (Polymorphism)**

Bir metodun farklı nesnelerde veya farklı ortamlarda farklı sonuçlar üretmesidir. Mesela “konuşmak” insan sınıfına ait bir metottur.

**Resim 1.6: Çok biçimlilik (Polymorphism) şeması**

Ancak bu metot ait olduğu insan sınıfının örneğine göre farklı sonuçlar üretir. Arkadaşımızla konuşurken farklı bir tonda, patronumuzla konuşurken farklı bir tonda konuşuruz, diğer bir ifadeyle bu metodun birden fazla biçimi vardır.

**1.2. Net Framework**

NET Framework, Microsoft tarafından geliştirilen, açık İnternet protokolleri ve standartları üzerine kurulmuş komple bir uygulama geliştirme platformudur.

Bu platform, işletim sisteminden ve donanımdan daha üst seviyede taşınabilir olarak tasarlanmıştır. Özet olarak bakıldığında önümüzde duran .NET Framework :

* İnterneti hedef alan bileşen setiyle,
* Programcıları özgürleştiren, diller arası etkileşime olanak tanıyan bağımsız dil mimarisiyle,
* Donanım-bağımsız ara seviye derleme ve güvenli talimat icraatını güvence altına alan çalışma zamanı ortamıyla,
* Web uygulamalarına getirdiği özgün ve radikal yaklaşımıyla,
* Zengin ve mükemmel organize edilmiş sınıf kütüphanesi ve dokümantasyonu ile
* Sınırları ortadan kaldıran web servisi desteğiyle, ve daha da sayabileceğimiz birçok özelliğiyle benzetildiği teknolojileri geride bırakan inkâr edilemeyecek seviyede özgün bir platformdur.

**1.2.1. Bu Platformu Nasıl Edineceğim?**

.NET platformu iki ayrı kurulum dosyası halinde, ücretsiz sunuluyor:

**1.2.2. NET'in Çekirdek Kurulumu (21MB)**

Bu kurulum ile bilgisayarınızda bir metin editörü açıp kodunuzu yazabilir, gelen derleyiciler ile derleyip uygulamanızı sınayabilirsiniz. Ancak bu kurulum, geliştiriciler için değil, daha çok dağıtım sonrası uygulamanızın çalışacağı sistemler için uygundur.

**1.2.3. NET Framework SDK (134 MB)**

SDK yani Software Development Kit, çekirdek kurulumu içeriğinden daha kapsamlı bir kurulumdur. Peki, 134 MB’lik kurulumda fazladan neler var: Faydalı araçlar, hata ayıklayıcılar, SQL Server’ın hafif versiyonu MSDE, hızlıca başlamanızı temin edecek QuickStart Tutorials, örnek uygulamalar ve devasa dokümantasyon.

**1.2.4. Hangi İşletim Sistemlerinde Çalışıyor?**

Şu anda. NET Framework istemci uygulamaları Windows 98/Me/NT4/2000/XP/Server 2003 işletim sistemlerinde çalışabiliyor. Ancak sunucu davranışı için Windows 2000 (SP2 ve üstü), Windows XP veya Windows Server 2003 gerekiyor. Windows Server 2003’te Net Framework kurulmuş oluyor zaten, ayrıca yüklemenize gerek yok.

**1.2.5. .Net Framework’ün Yapısı**

Java’dan önce, geliştirilen yazılımlar direkt olarak makine koduna çevrilirdi. Java ile program kodu önce byte code’a çevrilir. JVM (Java virtual machine) bu kodu işletim sisteminin istediği koda çevirir.

.Net içinde çalışma mantığı benzerdir .NET kodu ilk önce IL’ ye (Intermediate Language-Ara dil-veya MSIL) derler, bu IL kodu çalıştırılmak istendiğinde CLR, JIT derleyicilerini kullanarak kodu makine diline çevirir.



**Resim 1.7: .NET Framework’ün bileşenlerini gösteren bir başka resim**

**1.3. MSIL (Microsoft Intermediate Language)**

Normalde bir program derlendiğinde doğrudan makine koduna çevrilirken, .NET uyumlu bir dil ile derleme yapıldığında program kodu makine koduna değil de MSIL (Microsoft Intermediate Language)'e çevrilir.

MSIL işlemciden bağımsız komut setinden oluşmaktadır. Bu komut seti içerisinde nesnelerin yüklenmesi, depolanması ve initialize (başlatılması) edilmesini sağlayan komutların yanı sıra aynı zamanda nesneler üzerinde metot çağrımını sağlayan komutlar da yer almaktadır. .NET Framework ile yazılmış bir kaynak kodun derlenmesi sonucu MSIL kodu oluşmaktadır. MSIL, NET dilleri arasında bir bütünleşme sağlamaktadır. MSIL kodun çalıştırılmasından önce var olan MSIL kodu, doğal makine koduna (native code) dönüştürülmektedir. Burada interpretation dediğimiz yorumlama işlemi yapılmaz.



**CLR (Common Language Runtime):** Ortak Dil Çalışma Platformu olarak da adlandırılan CLR, makine diline çevrilmiş bu kodu önbellekte tutar, bu performans artışına sebep olurken diğer taraftan sistem hafızasında küçümsenmeyecek yer işgal eder.

* CLR, .NET altyapısında programların çalışmasını kontrol eden ve işletim sistemi ile programımız arasında yer alan arabirimdir. Normalde yazılan kodlar makine diline çevrilir ve işletim sistemi ile direkt bağlantı kurup çalışırdı.
* Eğer platformdan bağımsız bir ortam istiyorsak, ihtiyaç duyulan şey CLR dir, hangi platformda iseniz (Linux, Mac, Windows) CLR bu noktada devreye girer ve. NET programlarını farklı platformlarda işletim sistemine göre çalıştırır.

**1.4. Assembly Kavramı**

.Net'te derlenen sınıf kütüphaneleri (class library 'ler) ismine Assembly denilen dll dosyalarını oluşturur. Bu anlamda bir Assembly, bir ya da daha fazla fiziksel dosyayı barındıran bir koleksiyondur. Bu dosyalar; assembly içerisindeki derlenmiş sınıfların kodları, resimler, resource dosyaları ve diğer binary dosyalardır.

Assembly dosyaları .dll ya da .exe uzantılı olabilir. Assembly’lerin en önemli özelliği registry'i kullanmamasıdır.

**1.5. JIT Derleyiciler (Just in Time)**

C# ile IL’ ye derlediğimiz programı çalıştırırken JIT derleyicileri devreye girer. Bu derleyiciler programın çalıştırıldığı sistemin ve işlemcinin anlayabileceği makine kodunu oluşturur.

Windows ortamı için 3 çeşit JIT mevcuttur.

* **Normal JIT:** IL kodu makine koduna çevrilirken default (varsayılan) olarak kullanılan derleyicidir. IL kodunu orijinal makine koduna çevirir ve önbellekte tutar. Örneğin; program içindeki derlenmiş bir metot program akışı içinde tekrar çağrılırsa önbellekten çekilir.
* **Pre-JIT:** Tüm program kodunu makine koduna çevirip sonra çalıştıran JIT. fazla hafıza gerektirir. Programın daha hızlı çalışmasını sağlar.
* **Eco JIT:** Kısıtlı hafıza ve önbellekli sistemlerde. NET programlarının daha iyi çalışmalarını sağlamak için kullanılan derleyicidir.

**1.6. The Base Class Library - Temel Sınıf Kütüphaneleri**

.NET Framework’te programcıların işlerini kolaylaştırmak için birtakım hazır kütüphaneler vardır fakat C# dilinde hazır kütüphaneler mevcut değildir. Temel sınıf kütüphaneleri, uygulamalarınızı geliştirirken size lazım olabilecek birçok servisi ve nesneyi içinde barındıran bir sistemdir.

Temel sınıf kütüphaneleri, namespaces (isim alanları)’ler içinde organize edilmişlerdir.

* **System isim alanı:** .NET çalışırken gerekli temel sınıfları içerir. Ayrıca diğer\_tüm sınıf kütüphaneleri de bunun içinde kümelenmiştir. System hiyerarşinin tepesinde bulunur.
* **System.Data** : Veri tabanı işlemleri için kullanılır.
* **System.Data.SqlClient**: SQL işlemleri için kullanılır.
* **System.Net** : HTTP ve ağ protokolleri için kullanılır.
* **System.Xml** : XML verileri ile çalışmak için kullanılır.
* **System.IO** : Dosyalara bilgi girişi, dosyadan bilgi okuma, I/O işlemleri için kullanılır.
* **System.Windows.Forms**: Windows tabanlı uygulamalarda kullanılan zengin grafik arabirimi kontrollerini içerir.

C Sharp Programlama Dili/İlk programımız

İlk programımız şu şekilde:

class ilkprogram

{

static void Main()

{

System.Console.WriteLine("Merhaba dünya!");

}

}

Bu program konsol ekranına Merhaba dünya! yazıp kapanır. Programı geçen derste anlatıldığı gibi derleyin, sonra da programımızı çalıştırmak için komut satırında kod dosyanıza verdiğiniz adı girip entera basın.

### İlk programımızın satır satır incelenmesi

* class ilkprogram satırıyla *ilkprogram* adında yeni bir sınıf oluştururuz. C#'ta yazdığımız her programın en az bir sınıf içermesi zorunludur.
* { veya } işaretleri herhangi bir sınıfın veya metodun içeriğini belirtmek için kullanılır. İlk { karakteriyle önceki satırda açtığımız ilkprogram adlı sınıfımızın içeriğine alınacak kodların başladığını, programın son satırındaki } karakteriyle de sınıfımızın içeriğine alınacak kodların sona erdiğini belirtiyoruz.
* static void Main() satırıyla sınıfımızın içine Main adlı bir metot yerleştirdik. Bu metodun adı mutlaka Main olmalı, yoksa programımız çalışmaz. Ayrıca bu metot mutlaka oluşturduğumuz sınıfın içinde olmalı. Yine { ve } karakterleriyle metodumuzun içeriğini belirledik. Dikkat ettiyseniz bir iç içe { ve } karakterleri söz konusu. Bu durumda koyulan ilk } karakteri son açılan { karakterini kapatır.
* Programımızın pratikte iş yapan tek kısmı ise System.Console.WriteLine("Merhaba dünya!"); satırı. Bu satırla ekrana Merhaba dünya! yazdırdık, peki nasıl yaptık? Bunun için .Net Framework kütüphanesindeki hazır bir metottan yararlandık. Bu metot System isim alanının altındaki Console sınıfında bulunuyor. Ve bu metodumuzun adı WriteLine ve konsol ekranına yazı yazdırmaya yarıyor. Parantezler arasındaki çift tırnaklar arasına alınan metni ekrana yazdırıyor. Satırın sonundaki ; karakterini ise { ve } karakterleri açıp kapatmayan bütün C# satırlarında kullanmamız gerekiyor.

### Programımızın ikinci versiyonu

using System;

class ilkprogram

{

static void Main()

{

Console.WriteLine("Merhaba dünya!");

}

}

Bu programımızın işlevsel olarak ilk programımızdan herhangi bir farkı yok. Yani ikisi de aynı şeyi yapıyor. Ancak kodda bir farklılık var. Using System; satırı oluşmuş ve System.Console.WriteLine("Merhaba dünya!"); satırındaki System kalkmış. using deyimi C#'ın isim alanı kullanma hakkı elde etmek için kullanılan bir anahtar sözcüktür. Yani aynı isim alanında kullanacağımız birden fazla metot varsa bu isim alanını using anahtar sözcüğüyle belirtmemiz son derece mantıklı.

### Programımızın üçüncü versiyonu

using System;

class ilkprogram

{

static void Main()

{

Console.WriteLine("Entera basın!");

Console.ReadLine();

Console.WriteLine("Entera bastınız!");

}

}

Bu programımız önce ekrana Entera basın! yazar. Kullanıcı entera bastığında da Entera bastınız! yazıp kendini kapatır. Console sınıfına ait olan ReadLine metodu programımızın kullanıcıdan bilgi girişi için beklemesini sağlar, yani programımızı entera basılana kadar bekletir. Kullanıcı entera bastığında da diğer satıra geçilir. Dikkat ettiyseniz ReadLine metodunda parantezlerin arasına hiçbir şey yazmıyoruz. C#'ta bazı metotların parantezleri arasına birşeyler yazmamız gerekirken, bazı metotlarda da hiçbir şey yazılmaması gerekir.

C Sharp Programlama Dili/Değişkenler

Program yazarken her zaman sabit verilerle çalışmayız, çoğu zaman programımızda bir verinin kullanıcının davranışına göre değişmesi gerekir. Kullanıcıdan bir metin alıp bunu ekrana yazdıran bir program buna örnek verilebilir. Değişken kısaca bellek gözeneklerinin programlamadaki karşılıklarıdır.

### C#'ta değişken tanımlama

Çoğu programlama dilinde değişkenler kullanılmaya başlanmadan önce tanımlanırlar. Aşağıdaki şekli inceleyiniz.

[](http://tr.wikibooks.org/wiki/Dosya:Variable_definition_in_c_sharp.gif)

Yukarıdaki şekilde C#'ta değişken tanımlamanın nasıl yapıldığı anlatılmıştır. Değişken türü bellekte ayrılan gözeneğin büyüklüğünü belirtmemizi sağlar. Değişken adı da bu gözeneğe verdiğimiz adı belirtir. Doğal olarak bu gözenekteki veriye erişmek istediğimizde veya bu gözenekteki veriyi değiştirmek istediğimizde bu adı kullanacağız. Yukarıdaki şekilde -2,147,483,648 ile 2,147,483,647 arasında (sınırlar dâhil) bir değer tutabilen ve adı "ad" olan bir bellek gözeneği oluşturduk.

### Değişkenlere değer atama

Çoğu programlama dilinde değişkenler tanımlandıktan sonra direkt olarak programda kullanılabilirler. Ancak C#'ta değişkeni tanımladıktan sonra ayrıca bir de ilk değer atamak zorundayız. Aksi bir durumda değişkeni programımız içinde kullanamayız. Değişkenlere değer atama şöyle yapılır:

ad=5;

Burada ad değişkenine 5 değerini atadık. Bu en basit değer atama yöntemidir. Ayrıca şunlar da mümkündür:

int a=5;

int b, c, d, e;

int f=10, g, m=70;

Birinci satırda tanımlama ve değer vermeyi aynı satırda yaptık. İkincisinde aynı türden birden fazla değişken tanımladık. Üçüncü satırda ise tanımladığımız değişkenlerin bazılarına değer verirken bazılarına vermedik.

### Değişken türleri

Yukarıda değişken tanımlarken değişken türü için int kullanmıştık. C#'ta bunun gibi farklı kapasitelere sahip bir hayli daha değişken türü vardır. Ayrıca bazı değişken türleri sayısal, bazıları da metinseldir. Sayısal türler aşağıdaki tabloda listelenmiştir:

|  |  |  |  |
| --- | --- | --- | --- |
| **Tür** | **Boyut** | **Kapasite** | **Örnek** |
| byte | 1 bayt | 0, ..., 255 (tam sayı) | byte a=5; |
| sbyte | 1 bayt | -128, ..., 127 (tam sayı) | sbyte a=5; |
| short | 2 bayt | -32768, ..., 32767 (tam sayı) | short a=5; |
| ushort | 2 bayt | 0, ..., 65535 (tam sayı) | ushort a=5; |
| int | 4 bayt | -2147483648, ..., 2147483647 (tam sayı) | int a=5; |
| uint | 4 bayt | 0, ..., 4294967295 (tam sayı) | uint a=5; |
| long | 8 bayt | -9223372036854775808, ..., 9223372036854775807 (tam sayı) | long a=5; |
| ulong | 8 bayt | 0, ..., 18446744073709551615 (tam sayı) | ulong a=5; |
| float | 4 bayt | ±1.5\*10-45, ..., ±3.4\*1038 (reel sayı) | float a=5F; veya float a=5f; |
| double | 8 bayt | ±5.0\*10-324, ..., ±1.7\*10308 (reel sayı) | double a=5; veya double a=5d; veya double a=5D; |
| decimal | 16 bayt | ±1.5\*10-28, ..., ±7.9\*1028 (reel sayı) | decimal a=5M; veya decimal a=5m; |

Dikkat ettiyseniz bazı değişken türlerinde değer atarken değerin sonuna bir karakter eklenmiş, bu değişken türlerindeki değişkenlere değer atarken siz de bunlara dikkat etmelisiniz. Sıra geldi metinsel türlere:

|  |  |  |  |
| --- | --- | --- | --- |
| **Tür** | **Boyut** | **Açıklama** | **Örnek** |
| char | 2 bayt | Tek bir karakteri tutar. | char a='h'; |
| string | Sınırsız | Metin tutar. | string a="Ben bir zaman kaybıyım, beni boşver hocam"; |

String türüne ayrıca char ve/veya string sabit ya da değişkenler + işaretiyle eklenip atanabilir. Örnekler:

char a='g';

string b="deneme";

string c=a+b+"Viki"+'m';

C#'ta hem metinsel hem de sayısal olmayan türler de vardır:

#### bool

Koşullu yapılarda kullanılır. Bool türünden değerlere true, false veya 2<1 gibi ifadeler örnek verilebilir. Örnekler:

bool b1=true;

bool b2=false;

bool b3=5>4;

#### object

Bu değişken türüne her türden veri atanabilir. Örnekler:

object a=5;

object b='k';

object c="metin";

object d=12.7f;

Aslında C#'taki bütün değişken türleri object türünden türemiştir. Bu yüzden object türü diğerlerinin taşıdığı bütün özellikleri taşır. Ancak şimdilik bunu düşünmenize gerek yok. Bu, nesneye dayalı programlamanın özellikleriyle ilgili.

### Değişkeni programımız içinde kullanma

Şimdiye kadar değişkenleri tanımlayıp ilk değer verdik. Şimdi değişkenleri programımızda kullanmanın zamanı geldi. Bir örnek:

using System;

class degiskenler

{

static void Main()

{

byte a=5;

Console.WriteLine(a);

}

}

Burada a değişkeninin değerini ekrana yazdırdık. Başka bir örnek:

using System;

class degiskenler

{

static void Main()

{

byte a=5;

byte b=8;

Console.WriteLine(a+b);

}

}

Bu programda iki değişkenimizin değerlerinin toplamını ekrana yazdırdık.

using System;

class degiskenler

{

static void Main()

{

string a="Viki", b="kitap";

Console.WriteLine(a+b);

}

}

Bu programda aynı satırda iki tane string değişkeni tanımladık ve değer verdik. Bu değişkenlerin değerlerini WriteLine metoduyla ekrana yan yana yazdırdık. WriteLine metodu + işaretini gördüğünde sayısal değişken ve değerleri toplar, string türünden değişken ve değerleri yan yana yazar, char türünden değişken ve değerlerin Unicode karşılıklarını toplar. Ancak tabii ki + karakterinin ayırdığı değişken veya değerler char ile stringse char karakterle string metni yan yana yazar.

using System;

class degiskenler

{

static void Main()

{

string a;

a=Console.ReadLine();

Console.WriteLine(a+" metnini yazdınız.");

}

}

Sanırım şimdiye kadar yazdığımız en gelişmiş program buydu. Bu program kullanıcıdan bir metin alıp bunu ekrana "... metnini yazdınız." şeklinde yazıyor. Geçen derste ReadLine metodunu kullanıcı entera basana kadar programı bekletmek için kullanmıştık. Aslında ReadLine metodunun en yaygın kullanımı kullanıcının bilgi girişi yapmasını sağlamaktır. Dikkat ettiyseniz programımızda kullanıcının girdiği bilgi a değişkenine atanıyor. Sonra da WriteLine metoduyla ekrana bu değişken ve " metnini yazdınız." metni yan yana yazdırılıyor. Burada asıl önemsememiz gereken şey Console.ReadLine() ifadesinin string türünden bir değer gibi kullanılabilmesidir. C#'ta bunun gibi birçok metot bir değer gibi kullanılabilir. Tahmin edebileceğiniz üzere WriteLine gibi birçok metot da bir değer gibi kullanılamaz. Başka bir örnek:

using System;

class degiskenler

{

static void Main()

{

int a, b;

a=20;

b=a;

Console.WriteLine(b);

}

}

Bu programda da görebileceğiniz gibi değişkenlere değer olarak başka bir değişkeni atayabiliriz. Ancak değerini atadığımız değişkene daha önceden bir değer atanmış olması gerekiyor. Burada b değişkenine a değişkeninin değeri atanıyor. Ancak aşağıdaki gibi bir kullanım kesinlikle hatalıdır.

using System;

class degiskenler

{

static void Main()

{

Console.ReadLine()="metin";

string a=Console.ReadLine();

Console.WriteLine(a);

}

}

Burada Console.ReadLine() ifadesi bir değişkenmiş gibi kullanılmaya çalışılıyor, ancak hatalı. Çünkü Console.ReadLine() ifadesi yalnızca bir değermiş gibi kullanılabilir.

### Değişken adlandırma kuralları

Şimdiye kadar değişkenlere ad, a veya b gibi basit adlar verdik. Ancak aşağıdaki kuralları ihlal etmemek şartıyla değişkenlere istediğiniz adı verebilirsiniz.

* Değişken adları boşluk, simge içeremez.
* Değişkenler bir numerik karakterle başlayamaz.
* C#'ın diğer bütün komut, metot ve benzerlerinde olduğu gibi değişken adlarında büyük-küçük harf duyarlılığı vardır. Yani degisken isimli bir değişkenle Degisken isimli bir değişken birbirinden farklıdır.
* Değişken adları Türkçe karakterlerden(ğ,ü,ş,ö,ç,ı) oluşabilir.

### Sık yapılan hatalar

C#'ta değişkenlerle ilgili sık yapılan hatalar şunlardır:

* Aynı satırda farklı türden değişkenler tanımlamaya çalışma. Örneğin aşağıdaki örnek hatalıdır:

int a, string b;

* Değişkene uygunsuz değer vermeye çalışma. Örnek:

int a;

a="metin";

* Değişkeni tanımlamadan ve/veya değişkene ilk değer vermeden değişkeni kullanmaya çalışma. Aşağıdaki örnekte iki değişkenin de kullanımı hatalıdır.

using System;

class degiskenler

{

static void Main()

{

int b;

Console.WriteLine(a);

Console.WriteLine(b);

}

}

* Değişken tanımlaması ve/veya değer vermeyi yanlış yerde yapma. Örnek:

using System;

class degiskenler

{

int a=5;

static void Main()

{

Console.WriteLine(a);

}

}

Diğer using dışındaki bütün komutlarda da olduğu gibi değişken tanım ve değer vermelerini de Main bloğunun içinde yapmalıyız.

* Bazı değişken türlerindeki değişkenlere değer verirken eklenmesi gereken karakteri eklememek. Örnek:

using System;

class degiskenler

{

static void Main()

{

float a=12.5;

Console.WriteLine(a);

}

}

* Ondalık sayıların ondalık kısmını ayırırken nokta (.) yerine virgül (,) kullanmak. Örnek:

using System;

class degiskenler

{

static void Main()

{

float a=12,5f;

Console.WriteLine(a);

}

}

* Metinsel değişkenlerle matematiksel işlem yapmaya çalışmak. Örnek:

using System;

class degiskenler

{

static void Main()

{

string a="1", b="2";

int c=a+b;

Console.WriteLine(a);

}

}

* Bir değişkeni birden fazla kez tanımlamak. Örnek:

using System;

class degiskenler

{

static void Main()

{

string a;

string a="deneme";

Console.WriteLine(a);

}

}

* Değişkenlere değer verirken yanlış şekilde değer vermek. Örnek:

using System;

class degiskenler

{

static void Main()

{

string a=deneme;

Console.WriteLine(a);

}

}

### Sabit değişkenler

Programımızda bazen değeri hiç değişmeyecek değişkenler tanımlamak isteyebiliriz. Örneğin pi isimli float türünden bir değişken tanımlayıp buna 3.14 değerini verip programımızda pi sayısına ihtiyaç duyulduğunda bu değişkeni kullanabiliriz. Sabit değişkenlerin normal değişkenlerden farkı değişkeni değiştirmek istediğimizde ortaya çıkar, sabit olarak belirtilen değişkeni değiştirirsek derleyici hata verip programımızı derlemez. Bu daha çok uzun program kodlarında işe yarayabilir. Ayrıca sabit değişkenlere tanımlandığı satırda değer vermeliyiz. Herhangi bir değişkeni sabit olarak belirtmemiz için değişken türünden önce const anahtar sözcüğü kullanılır. Örnekler:

const int a = 5;

const string b ="deneme";

const char c ='s';

Aşağıdaki gibi bir durum, değişkene tanımlandığı satırda değer verilmediği için hatalıdır.

const int a;

a=5;

Sabit ifadelere değer olarak sabit, sabit değişken ya da sabit ve/veya sabit değişkenlerden oluşan matematiksel ifadeler verilebilir. Örnek:

const int a=5;

const int b=a+7;

const int c=a\*b;

Aşağıdaki gibi bir durum hatalıdır.

int a=5;

const int b=a+8;

### Escape sequence kullanımı

Bir string sabite özel karakterler eklemek için veya WriteLine ile bir özel karakter yazdırabilmek için escape sequence kullanılır. Örnekler:

string ad="Deneme\"Deneme";

Console.WriteLine(ad);

Bu satırda " karakterinin ad stringinin içine koyulmasını sağladık. Yukarıdaki kod ekrana Deneme"Deneme yazar. Yani stringimizin kendi içine almasını istediğimiz illegal karakterden hemen önce \ işareti koyuyoruz. Başka bir örnek:

string yol="Windows\\Program Files";

Burada bir illegal karakter olan \ karakterinin başına tekrar \ koyarak stringin bir tane \ almasını sağladık.

string yol=@"Windows\Program Files";

Burada stringe tırnak içerisindeki metin olduğu gibi aktarılır, ancak doğal olarak " karakterinde işe yaramaz.

Console.WriteLine("Satır\nYeni satır\nYeni satır");

Örnekte de gördüğünüz üzere C#'ta \n yeni satır yapmak için kullanılır. String sabitlerde kullanılabilir.

### Ek bilgiler

Burada değişkenlerle ilgili olan ancak herhangi bir başlıkta incelenemeyecek olan önemli bilgiler bulunmaktadır. Bunların bazıları kendiniz çıkarabileceğiniz mantıksal bilgilerdir.

* ReadLine() metodunun tuttuğu değer string türündedir.
* WriteLine() metodunda parantezler arasına girilen değer object türünden olmalıdır. Yani herhangi bir türden değer yazılabilir.
* Bütün değişkenler bir değermiş gibi kullanılabilir ancak değerler değişkenmiş gibi kullanılamaz.
* Eğer bir değişkeni tanımlamış ve/veya değer vermiş, ancak programımızın hiçbir yerinde kullanmamışsak derleyici hata vermez, sadece bir uyarı verir.
* "deneme"+"yalnızlık"+'d'+'m' gibi bir ifade aslında string türünden bir sabittir.
* (3+8/9)\*6 gibi bir ifade aslında byte, sbyte, short, ushort, int, uint, long, ulong veya double türünden bir sabittir. Yani bu türden değişkenlere atanabilir.
* Aslında C#'ta sabitlerin de türleri vardır. Mesela:
  + "5" string türünden
  + '5' char türünden
  + 5f float türünden
  + 12.7 veya 5d veya 5D double türünden (C# harfi olmayan ondalıklı sayıları double sayar.)
  + 5m decimal türünden
  + 5 ise teorik olarak diğer kalan sayısal türler türünden bir değişkendir. Ancak C# herhangi bir ayırt edici özelliği bulunmayan tam sayı sabitleri int türünden sayar.
* 5+"deneme"+6.7f gibi bir ifade aslında object türünden bir sabittir.
* C#'ta herhangi bir sabit ya da değişkenin türünü anlamak için x.GetType() metodu kullanılır. Örnek:

Console.WriteLine(5.GetType());

Console.WriteLine(14.4.GetType());

Console.WriteLine("deneme".GetType());

byte a=2;

Console.WriteLine(a.GetType());

Ancak GetType() metodunun tuttuğu değer Type türündedir, dolayısıyla herhangi bir string türündeki değişkene vs. atanamaz, ayrıca da GetType() metodu değişkenin CTS'deki karşılığını verir. Değişkenlerin CTS karşılıkları bir sonraki dersimizin konusudur.

* Sabitin diğer adı değerdir.
* a=b ifadesi atama işlemidir. b'nin değeri a'ya atanır. b sabit ya da değişken olabilir. Ancak a kesinlikle değişken olmalıdır.