

Visual Studio.Net -C#

8. HAFTA

Göstericiler, Ön işlemciler, Temel Giriş Çıkış İşlemleri (I/O)

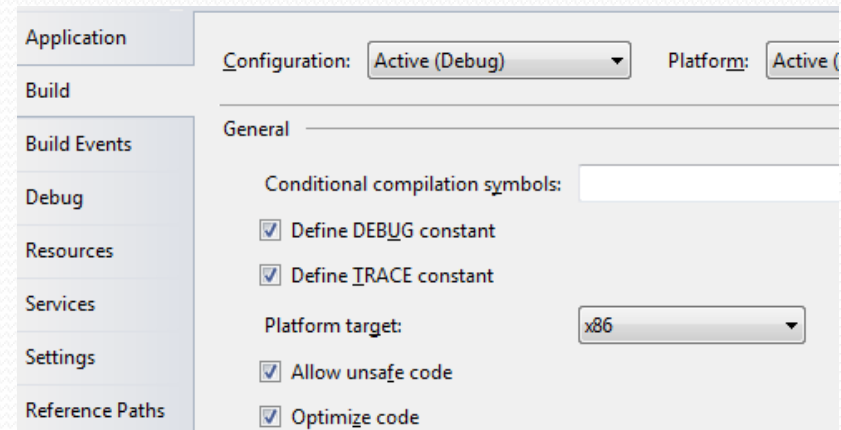
C#’ta Gösterici(Pointer) Kullanımı ve Emniyetsiz(unsafe) Kod

- Her ne kadar .NET platformunda bellek yönetimi gereksiz bilgi toplayıcısı(Garbage collector) dediğimiz mekanizma ile otomatik olarak sağlansa da bazen **direk belleğe erişmemiz** gerekebilir.
- Bu durumda **işaretçiler(pointer)** kullanılır. .NET’in alt yapısında pointerlar sıklıkla kullanılırken, pointer kullanımının programdaki görünmeyen hatalara sıkça yol açtığından dolayı programcılar için gizlenmiştir.
- pointer kullanımı birçok sıkıntıya yol açmaktadır. Bunlardan en önemlileri kullanım zorluğu ve bulunması zor hatalara yol açabilmesidir.
- C# ‘ta pointer kullanılacağı zaman kod blokları **unsafe** anahtar kelimesi kullanılarak işaretlenmelidir. Aksi halde program derleme esnasında hata verecektir.

C#'ta Gösterici(Pointer) Kullanımı ve Emniyetsiz(unsafe) Kod

- Bir sınıfı unsafe olarak belirtirsek o sınıf içinde gösterici kullanabiliriz.
- **unsafe class** Sınıf { ... }
- Herhangi bir metodun içinde bir unsafe bloğu oluşturarak o bloğun içinde gösterici kullanabiliriz.
- **int** Metot() { **unsafe** { ... } }
- Normal bir metodu unsafe olarak belirterek o metot içinde gösterici kullanabiliriz.
- **unsafe int** Metot() { ... }
- Özellikleri unsafe olarak belirterek yeni bir gösterici oluşturabiliriz. Bu durumda sınıfı unsafe olarak belirtmeye gerek kalmaz. Ancak bir metot içinde bir gösterici oluşturabilmek için ya o metodun unsafe olarak bildirilmesi ya da bir unsafe bloğunun açılması gerekir. Ayrıca içinde gösterici kullandığımız bir kaynak kod dosyası normal yollarla derlenemez. İçinde gösterici kullandığımız bir kaynak kod dosyamızı aşağıdaki iki yoldan biriyle derleriz.

- `csc /unsafe KaynakKod.cs` veya
- `csc -unsafe KaynakKod.cs`



C#’ta Gösterici(Pointer) Kullanımı ve Emniyetsiz(unsafe) Kod

- Çok istisnai durumlar dışında C # ‘ta pointerlara ihtiyaç duyulmaktadır.
- Bu durumları aşağıdaki gibi sıralayabiliriz.
- **Geriye uyumluluk(Backward compatibility):** COM ve WinAPI ‘deki fonksiyonlar gibi pointerları sık kullanan fonksiyonları C# ile çağırabilmek için parametre olarak gösterici alan fonksiyonlara gösterici gönderdiğimiz durumlarda kullanabiliriz.
- **Performans:** Belleğe direk eriştiği için performansı aşırı derecede yükselttiği için kullanılır.
- **Alt seviye İşlemler:** donanım arayüzleri ile direk ilişkili olduğu için kullanımları mecburi gibidir.
- **Hata Ayıklayıcı Programlar:** hata ayıklayıcı programları yazabilmek için belleğe direk erişim sağlanmalıdır. Bu durumda pointer kullanımı şarttır.

Gösterici bildirimi

- Gösterici bildirimi ilgili yapı adının sonuna * işareti koyularak yapılır. Örnekler:
- `char*` gosterici1;
- `int*` gosterici2;
- Bu göstericilerden birincisi bir char türünden nesnenin (değişkenin), ikincisi de bir int türünden nesnenin adresini tutabilir.
- Birden fazla aynı türde gösterici oluşturmak için normal veri türlerinde olduğu gibi virgül kullanılabilir.
- Örnek:
- `int*` gosterici1, gosterici2;
- Göstericileri sabit ifadeler ile de bildirebiliriz. Örnek:
- `double*` gosterici=(`double*`)123456;
- Burada bellekteki 123456 adresi gosterici göstericisine atanıyor. Ancak biz 123456 adresinde ne olduğunu bilmediğimiz için bu tür bir kullanım son derece tehlikelidir.

Gösterici bildirimi

- **& operatörü:**

- Bu operatör adres operatörüdür. & işlemi önünde yer aldığı değişkenin adresini döndürür. Örneğin aşağıda int türünden bir pointer' a daha önceden tanımlanan int türünden bir nesnenin adresi atanmaktadır.

- **int sayi=99; int* p; p=&sayi;**

- *** Operatörü:**

- * operatörü bir adresteki bilgileri elde etmek ve değiştirmek için kullanılır. Aşağıda int türünde bir adres olan *sayi* nesnesinin adresine * operatörü ile yeni değer atanıyor.

- **int sayi=99; int* p; p=&sayi; *p=100;**

- **Notlar:**

- **1-)** Henüz adresi olmayan bir göstericiye değer atanamaz. Örneğin şu kod hatalıdır:

- **int* gosterici; *gosterici=10;**

- **2-)** Ancak henüz bir değeri olmayan bir nesnenin adresi alınabilir. Yani şu kod hata vermez:

- **int a; int* ptr=&a;**

Göstericiler arasında tür dönüşümü

- Göstericiler arasında bilinçli tür dönüşümü yapılabilir. Ancak dikkatli olunmalıdır. Çünkü büyük türün küçük türe dönüşümünde veri kaybı yaşanabilir.
- Küçük türün büyük türe dönüşümünde de küçük tür bellekte kendine ait olmayan alanlardan da alır. Dolayısıyla her iki durumda da istemediğimiz durumlar oluşabilir. Göstericiler arasında bilinçsiz tür dönüşümü ise imkansızdır. Örnek bir program:
- `using System;`
- `class Gostericiler {`
- `unsafe static void Main()`
- `{ char c='A'; int i=80; char* cg=&c;`
- `int* ig=&i; cg=(char*)ig; Console.WriteLine(*cg); }`
- `}`
- Bu program sonucunda ekrana P yazılacaktır. (80'in Unicode karşılığı P'dir.) Bu programda herhangi bir veri kaybı gerçekleşmedi. Çünkü 80 char türünün kapasitesindeydi.

Göstericiler arasında tür dönüşümü

- Göstericiler yapı nesnelerinin bellekteki adreslerini tutarlar. İşte bu adresi elde etmek için ilgili gösterici bir tam sayı türüne dönüştürülmelidir. Örnek:
- `using System;`
- `class Gostericiler`
- `{ unsafe static void Main()`
- `{ int i=80; int* ig=&i; uint adres=(uint)ig; Console.WriteLine("{0:X}",adres); }`
- `}`
- Bu program sonucunda benim bilgisayarım 13F478 çıktısını verdi. Ancak bu sizde değişebilir. Çünkü işlemcinin verileri nereye koyacağını hiç birimiz bilemeyiz. Ayrıca adresi ekrana yazdırırken formatlama yapıp 16'lık düzende yazdığının da farkında olmalısınız. Çünkü bellek adresleri genellikle 16'lık düzendeki sayılarla ifade edilir.
- Göstericilerin adreslerini elde ederken uint türüne dönüşüm yapmak zorunlu değildir. İstedığınız tam sayı türüne dönüşüm yapabilirsiniz. Ancak uint, bir bellek adresini tam olarak tutabilecek en küçük veri tipi olduğu için ilgili göstericiyi uint türüne dönüştürmeniz tavsiye edilir.

Göstericiler

- **void göstericiler**
- void tipinden göstericilere her türden adres atanabilir. void türü nesnelerdeki object türüne benzer. Örnek kod
- **int*** gosterici1;
- **void*** gosterici2;
- gosterici2=gosterici1;
- **Sizeof Operatörü:**
- Temel veri türlerinin ve yapılarının bellekte ne kadar yer kapladıklarını *sizeof* operatörü ile bulabiliriz. Örneğin
- sizeof(int)=4;
- **using** System;
- **class** Gostericiler {
- **struct** yapi
- { **public int** Ozellik1; **public int** Ozellik2; }
- **unsafe static void** Main()
- { Console.WriteLine(sizeof(yapi)); }
- }
- Bu program 8 çıktısını verir.

Gösterici aritmetiği

```
• using System;
• class pointerkullanimi
• {
•     unsafe public static void Main()
•     { int* sayi1=(int*) 100;
•       char* sayi2=(char*) 100;
•       byte* sayi3=(byte*) 100;
•
•       sayi1+=2; sayi2+=5; sayi3+=6;
•
•       Console.WriteLine((uint) sayi1);
•       Console.WriteLine((uint) sayi2);
•       Console.WriteLine((uint) sayi3);
•     }
• }
```

Programın çıktısı:

108

110

106

Gösterici aritmetiği

- Göstericiler yalnızca tam sayılarla matematiksel işleme girerler. +, -, --, ++, -= ve += operatörleri göstericilerle kullanılabilir. void göstericiler matematiksel işleme girmezler.
- İki gösterici birbirinden çıkarılabilir. Ancak bu durumda bir gösterici değil, long türünden bir nesne oluşur. Örnek:

```
using System;  
class Gosterici  
{  
    unsafe static void Main()  
    { int* g1=(int*)500;      int* g2=(int*)508;  
  
        long fark=g2-g1;      Console.WriteLine(fark);  
    }  
}
```

- Bu program sonucunda ekrana 2 yazılır.
- Çünkü göstericilerde çıkarma yapılırken şu formül kullanılır:

(Birinci göstericinin adresi - İkinci göstericinin adresi)/Ortak türün byte cinsinden boyutu
(508-500)/4=2-----→ int 4 byte yer kaplar

Önişlemci komutları- #define

- **#define** komutu bir sembol tanımlamak için kullanılır. Sembollerin tek başlarına bir anlamı yoktur
- #define ile sembol tanımlama program kodlarının en başında, (using deyimlerinden de önce) olmalıdır.
- Örnek program:
- #define ENGLISH
- #define TURKISH
- using System;
- class Onislemci { static void Main() { } }
-
- Bu programda ENGLISH ve TURKISH isimli iki sembol tanımlanmıştır. Diğer bütün önişlemci komutlarında olduğu gibi önişlemci komut parçaları arasında (örneğin #define ile TURKISH arasında) en az bir boşluk olmalıdır. Aynı sembolün tekrar tekrar tanımlanması hata verdirmez.

Önişlemci komutları-#undef

- **#undef**, #define ile tanımlanmış bir sembolü kaldırmak için kullanılır.
- Olmayan bir sembolü kaldırmaya çalışmak hata verdirmez. Programın herhangi bir satırında olabilir. Örnek program:
- #define ENGLISH
- using System;
- #undef ENGLISH
- class Onislemci
- {
- static void Main() { ... }
- }
- Sembollerin büyük ya da küçük harf olması zorunlu değildir. Ancak tamamen büyük harf okunurluğu artırdığı için tavsiye edilir. Sembollerin büyük-küçük harf duyarlılığı vardır.

Önişlemci komutları- #if , #endif

- **#if , #endif** komutları bir sembolün tanımlanmış olup olmamasına göre birtakım kodların derlenip derlenmemesinin sağlanması için kullanılır. Örnek program:
- `#define ENGLISH`
- `using System;`
- `class Onislemci { static void Main()`
- `{ Console.WriteLine("Programa hoş geldiniz.");`
- `#if ENGLISH Console.WriteLine("Bu program ENGLISH"); #endif }`
- `}`
- Bu program ekrana şu çıktıyı verecektir.
- **Programa hoş geldiniz. Bu program ENGLISH**
- Eğer programın başına `#define ENGLISH` satırını eklemeseydik program çıktısındaki ikinci satır olmayacaktı.

Önişlemci komutları- #if , #endif

- #if önişlemci komutuyla mantıksal operatörleri de kullanabiliriz. Örnek program:
- #define ENGLISH
- #define TURKISH
- using System;
- class Onislemci { static void Main()
- { Console.WriteLine("Programa hoş geldiniz.");
- #if ENGLISH && (!TURKISH) Console.WriteLine("Bu program ENGLISH"); #endif } }
- Bu program sonucunda ekrana yalnızca Programa hoş geldiniz. satırı yazılacaktır.
- Şimdi bu mantıksal operatör durumlarını topluca görelim:
#if ENGLISH && TURKISH ENGLISH ve TURKISH varsa
#if ENGLISH || TURKISH ENGLISH veya TURKISH varsa
#if ENGLISH && (!TURKISH) ENGLISH var ama TURKISH yoksa
#if ENGLISH && (TURKISH==false) ENGLISH var ama TURKISH yoksa
Benzer şekilde çoklu koşullandırmalar da mümkündür. Örnek:
#if ENGLISH && TURKISH && FRANSIZCA

Önişlemci komutları- #else

- **#else** önişlemci komutu C#'taki else ile aynı göreve sahiptir. Koşul sağlanmışsa bir kod bloğunun derlenmesini sağlayan #if komutu ile birlikte kullanılır. Örnek:
- #define ENGLISH
- using System;
- class Onislemci
- { static void Main()
- { Console.WriteLine("Programa hoş geldiniz.");
- #if ENGLISH Console.WriteLine("Bu program ENGLISH");
- #else Console.WriteLine("Bu program ENGLISH değil");
- #endif
- }
- }

Önişlemci komutları- #else

- #define ENGLISH
- **using** System;
- #if ENGLISH
- **class** Onislemci {
- **static void** Main()
- { Console.WriteLine("Bu program ENGLISH"); }
- }
- #else **class** Onislemci {
- **static void** Main()
- { Console.WriteLine("Bu program ENGLISH değil"); }
- }
- #endif

Önişlemci komutları- #elif

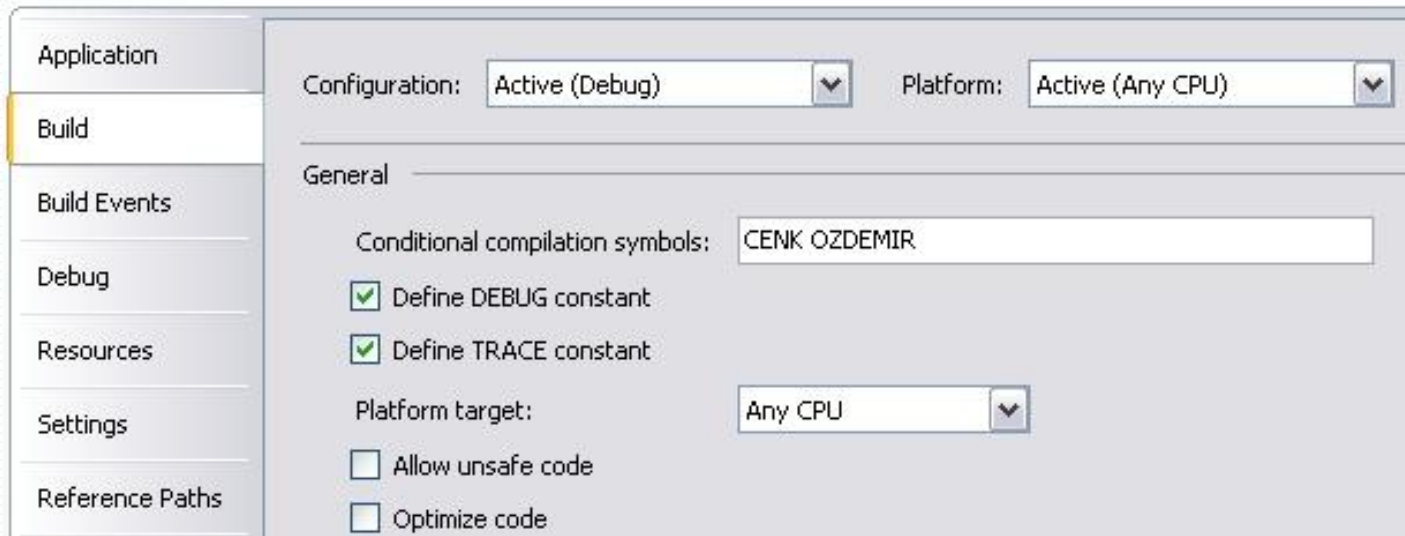
- **#elif**, C#'taki elseif 'in kısaltılmışıdır. Yani değilse bir de bu koşula bak anlamı verir.
Örnek program:
- `#define ENGLISH`
- `using System;`
- `class Onislemci {`
- `static void Main() {`
- `Console.WriteLine("Programa hoş geldiniz.");`
- `#if ENGLISH Console.WriteLine("Bu program ENGLISH");`
- `#elif TURKISH Console.WriteLine("Bu program TURKISH");`
- `#else Console.WriteLine("Bu program hem TURKISH, hem de ENGLISH değil");`
`#endif } }`
- #elif komutlarının sayısını bir #if-#endif aralığında istediğimiz kadar artırabiliriz.

Önişlemci komutları- #elif

- İç içe geçmiş #if-#elif-#else komutları da olabilir. Örnek program:
- #define ENGLISH
- using System;
- class Onislemci {
- static void Main() {
- #if ENGLISH
- #if TURKISH Console.WriteLine("Bu program hem ENGLISH hem TURKISH");
- #endif
- #elif TURKISH Console.WriteLine("Bu program sadece TURKISH");
- #endif } }

#define -1-Proje özelliklerinde koşul tanımlama

- #define önışlemci direktifi dışında, proje özelliklerinden faydalanarak da koşul tanımlaması yapılabilir.
- Bunun için proje özellikleri altındaki Build seçeneğinde, "Conditional compilation symbols" yazan yere derleme koşullarını söylemek gerekir. Birden fazla koşul tanımlaması yapılırken, koşullar arasında bir boşluk bırakılabilir, virgül ya da noktalı virgül konabilir.



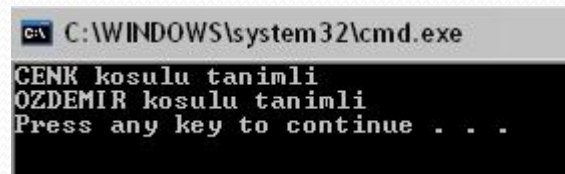
#define -1-Proje özelliklerinde koşul tanımlama

- using System;
- namespace OnIslemciDirektifleri

```
{  
    class Program  
    {  
        static void Main(string[] args)  
        {  
  
            #if CENK  
                Console.WriteLine("CENK kosulu tanimli");  
            #endif  
            #if OZDEMIR  
                Console.WriteLine("OZDEMIR kosulu tanimli");  
            #endif
```

• // Tanımlanan kosullar iki kelimededen olusamazlar.Dolayısıyla asagidaki kod hata üretir.
// #if CENK OZDEMIR
// Console.WriteLine("CENK OZDEMIR kosulu tanimli");
// #endif

```
}  
}
```



```
C:\WINDOWS\system32\cmd.exe  
CENK kosulu tanimli  
OZDEMIR kosulu tanimli  
Press any key to continue . . .
```

#define -2- Komut satırında koşul tanımlama

- Komut satırından csc.exe'yi kullanarak yaptığımız derleme sırasında da koşul tanımlama imkanımız vardır. Bunun için, derlenmesini istediğimiz kaynak kod dosyasının adını söyledikten sonra /define anahtarından faydalanmak gereklidir.

- Test.cs**

using System;

- class Program**

```
{
    static void Main(string[] args)
    {
        #if KOSUL1
            Console.WriteLine("KOSUL1 tanımlı");
        #endif

        #if KOSUL2
            Console.WriteLine("KOSUL2 tanımlı");
        #endif
    }
}
```

```
C:\>csc Test.cs /define:KOSUL1,KOSUL2
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:\>Test.exe
KOSUL1 tanımlı
KOSUL2 tanımlı

C:\>_
```

- Daha sonra kaynak kodumuzu komut satırından yandaki gibi derleyip çalıştırın

Önişlemci komutları-#error

- **#error** komutu derleyiciyi hata vermeye zorlamak için kullanılır.
- Örnek program:
- #define ENGLISH
- #define TURKISH
- using System;
- class Onislemci {
- static void Main() {
- #if ENGLISH && TURKISH
- #error Hem TURKISH hem ENGLISH aynı anda olamaz.
- #endif }
- }
- Bu program kodumuz derlenmeyecek, hata mesajı olarak da ekrana Hem TURKISH hem ENGLISH aynı anda olamaz. yazacaktır.


Önişlemci komutları-#warning

- **#warning**, komutu #error komutu ile aynı mantıkta çalışır. Tek fark ekrana hata değil, uyarı verilmesidir. Yani programımız derlenir, ama uyarı verilir. Örnek program:
- #define ENGLISH
- #define TURKISH
- using System;
- class Onislemci {
- static void Main() {
- #if ENGLISH && TURKISH
- #warning Hem TURKISH hem ENGLISH tavsiye edilmez.
- #endif }
- }

Önişlemci komutları

- **Örnek:**

- `using System;`
- `namespace OnIslemciDirektifleri`
`{`
 `class Program`
 `{`
 `static void Main(string[] args)`
 `{`
 `int sayi = 1 / 0;`
 `}`
 `static void Baglan()`
 `{ #error Baglan metodu henüz yazılmadı. }`
 `static bool KullaniciBul(int id)`
 `{ #error KullaniciBul metodu henüz yazılmadı. }`
 `static bool KullaniciSil(int id)`
 `{ #warning KullaniciSil metodu henüz yazılmadı. }`
 `}`
`}`



2 Errors 1 Warning 0 Messages	
	Description
1	#error: 'Baglan metodu henüz yazılmadı.'
2	#error: 'KullaniciBul metodu henüz yazılmadı.'
3	#warning: 'KullaniciSil metodu henüz yazılmadı.'

- Yukarıdaki kodu derlemeye çalıştığımızda Error List aşağıdaki gibi görünür.
- Burada dikkatinizi çekmek istediğim nokta, #error direktifini kullandığımızda, kodtaki diğer hataları ele alamıyor oluşumuzdur. Aslında kodumuz içerisinde üç tane hata mevcut (0'a bölmeye çalıştığımız satır, geriye bir şey döndürmeyen KullaniciBul(int id) ve KullaniciSil(int id) metotları) olmasına rağmen, yalnızca #error ve #warning direktiflerince üretilen hata ve uyarılar ele alınabilmektedirler.

Önişlemci komutları- #line

- **#line**, eğer programımızda bir hata var ve derleyici de (csc.exe) bu hatayı tespit etmişse bize bu hatanın hangi dosyanın hangi satırında olduğunu verir.
- İstersek #line komutuyla bu dosya adını ve satır numarasını değiştirebiliriz. Yani tabiri caizse derleyiciyi kandırabiliriz. #line komutu kendinden hemen sonra gelen satırın özelliklerini değiştirmek için kullanılır. Örnek program:
- **using** System;
- **class** Onislemci {
- **static void** Main()
- {
- **#line** 1 "YeniDosya.cs"
- **int** a="deneme"; }
- }
- Bu programda int türünden olan a değişkenine yanlış türde sabit atadık. Bu hatayı bilinçli olarak yaptık. Derleyici hatanın 1. satırda ve YeniDosya.cs dosyasında olduğunu iddia edecektir. Çünkü artık derleyici #line 1 "YeniDosya.cs" komutundan hemen sonra gelen satırı 1. satır ve YeniDosya.cs dosyası olarak tanıyor.

Önişlemci komutları- #line

- Örnek: Satır numaralarıyla oynamak

- using System;

- namespace OnIslemciDirektifleri
{ class Program

- {
static void Main(string[] args)
{ #line 55 "bizimDosya"
int sayi1 = 1 / 0; //10

- #line 10 "bizimDosya" //11
int sayi1; //12

- #line 80 "bizimDosya" //13

- //14
int sayi2 = 1 / 0; //15

- #line 1 "sizinDosya" //16
int sayi2; //17

- }
 - }
 - }

- Kod derlendikten sonra, hata oluşan satırlar 10,12,15 ve 17 olmasına rağmen aşağıdaki Error List görünür.

Error List			
4 Errors 0 Warnings 0 Messages			
	Description	File	Line
1	Division by constant zero	bizimDosya	55
2	A local variable named 'sayi1' is already defined in this scope	bizimDosya	10
3	Division by constant zero	bizimDosya	81
4	A local variable named 'sayi2' is already defined in this scope	sizinDosya	1

Önişlemci komutları- #line

- **Örnek:** Hata ayıklayıcısına yön vermek

- using System;

- namespace OnIslemciDirektifleri

```
{  
    class Program  
    {  
        static void Main(string[] args)  
        {  
            #line hidden  
            Console.WriteLine("Cenk");  
            Console.WriteLine("Özdemir");  
            #line default  
            Console.Write("C#");  
            #line hidden  
            Console.WriteLine(" Nedir?");  
        }  
    }  
}
```

- Örnekteki kod içerisindeki hatalar ayıklanırken (debug) adım adım ilerlediğimizde, "#line hidden" satırından sonraki satırlara geçilmez, "#line default" satırına gelindikten sonra ise normal davranışa geri dönlür.

Önişlemci komutları-#pragma warning

- C#'ta zaman zaman kodun çalışmasına engel teşkil etmeyen uyarılarla karşılaşırız. #pragma warning ifadesi ise bize tüm uyarıları ya da belli bir uyarıyı kapatıp/açma, kısaca uyarıları yönetme olanağı sunar.

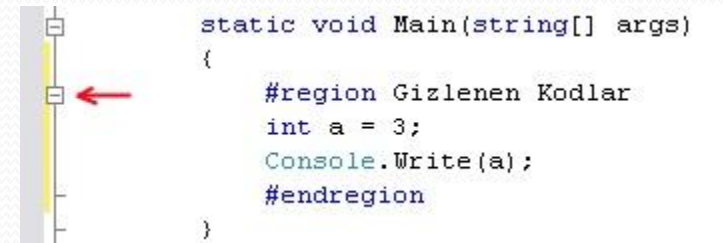
```
static void Main(string[] args)
{
    // 0168 kodlu uyarı, tanımladığımız değişkeni kullanmadığımız durumlarda verilir.

    #pragma warning disable 0169 // 0169 kodlu uyarıyı etkisiz hale getirir
        int i; !
    #pragma warning restore 0169 // 0169 kodlu uyarıyı aktif hale getirir
        int j; !
    #pragma warning disable 0168 // 0168 kodlu uyarıyı etkisiz hale getirir
        int k;
    #pragma warning restore 0168 // 0168 kodlu uyarıyı aktif hale getirir
        int l; !
    #pragma warning disable // Tüm uyarıları etkisiz hale getirir
        int x;
    #pragma warning restore // Tüm uyarıları aktif hale getirir
        int y; !
}
```

- Tüm uyarılarla ilgili seçenekleri, proje özelliklerinde (Project Properties) kod analiz (Code Analysis) seçeneği altında bulabilirsiniz.

Önişlemci komutları-#region / #endregion

- **#region / #endregion**, komutlarının, kodları Not Defteri'nde yazan bizler için pek bir önemi yok. Çünkü #region ve #endregion komutları kodları bölgelere ayırmaya yarıyor.
- Kod bölgesinin yanındaki + işaretine tıklayınca kodlar gösteriliyor, gösterilen kodların yanındaki - işaretine tıklayınca da kodlarımız gizleniyor. Bu komut Visual Studio veya Not Defteri'nden daha gelişmiş bir editör programı kullananlar için faydalı olabilir. Örnek:
- **using** System;
- **class** Onislemci {
- **static void** Main()
- { **#region** YeniAlan
- *//Bu kısmı yeni bir bölge yaptık*
- **#endregion**
- }
- }
- **Neden önişlemci komutları?**
- Önişlemci komutları genellikle belirli koşullara göre programımızın yalnızca bir kısmını derlemek için kullanılır. Örneğin bir programın hem İngilizce hem Türkçe versiyonunu aynı kaynak kodda bulunduruyorsak önişlemci komutlarını programın yalnızca Türkçe ya da yalnızca İngilizce versiyonunu derlemek amacıyla kullanabiliriz.



Input (Giriş)/Output (Çıkış)

Temel I/O işlemleri

Temel I/O işlemleri

- Programlama dilleri içerisinde yer alan önemli yapılardan biri de I/O sistemidir.
- I/O sistemi bilgisayarın çeşitli kaynaklarına erişmek için kullanılacak yollar anlamına gelir.
- Dosya kaydetme, ekrana ya da yazıcıya bilgi yazdırma, klavyeden bilgi girişi birer I/O işlemleridir.
- C# dilinde de I/O sistemi ile ilgili tüm sınıflar **System.IO** isim alanı altında bulunmaktadır.
- C# I/O sistemi akımlar (stream) üzerine kuruludur. Akımlar bir girdi ya da çıktı sistemi üzerinden byte düzeyinde bilgiyi okuyan sanal(soyut) birimlerdir.
- Stream sınıfı akımlar için gerekli temel metot ve özellikleri barındırır.

Directory sınıfı

- Directory sınıfının hiçbir özelliği yoktur, System.IO altında bulunur, sadece static metotlar içerir.
- **DirectoryInfo CreateDirectory (string adres):**
- adres ile belirtilen adreste bir klasör oluşturur ve bu klasör bilgilerini bir DirectoryInfo nesnesi olarak tutar. Programımızın çalıştığı klasörde bir klasör oluşturmak için sadece klasörün adını yazmak yeterlidir. Örnekler:
 - **Directory.CreateDirectory(@"C:\WINDOWS\deneme");**
 - Bu kod C:\WINDOWS altında deneme isimli bir klasör oluşturur.
 - **Directory.CreateDirectory("deneme");**
 - Bu kod programın çalıştığı klasörde deneme isimli bir klasör oluşturur.
 - **Directory.CreateDirectory(@"..\deneme");**
 - Bu kod programın çalıştığı klasörün bir üst klasöründe deneme isimli bir klasör oluşturur.
 - **Directory.CreateDirectory(@"..\..\deneme");**
 - Bu kod programın çalıştığı klasörün iki üst klasöründe deneme isimli bir klasör oluşturur. .. sayıları bu şekilde artırılabilir. Bu tür bir adres belirtme şekli bütün diğer metotlarda da geçerlidir. Ayrıca bu ve diğer bütün metotlarda da adres diye tarif ettiğimiz veriye dosya/klasörün adı da dâhildir.

Directory sınıfı

- **void Delete(string adres)**
- Belirtilen adresteki boş klasörü silmek için kullanılır. Başka bir kullanımı daha vardır.
- **void Delete(string adres, bool a)**
 - Bu metot ile eğer a true ise belirtilen adresteki klasör, içindeki bütün dosya ve klasörlerle birlikte silinir.
- **bool Exists(string adres)**
- Belirtilen adresteki klasörün olup olmadığını bool cinsinden tutar. Klasör varsa true, yoksa false döndürür.
- **string GetCurrentDirectory()**
- Çalışan programın hangi klasörde olduğunu verir. Örneğin Windows'taysak C:\WINDOWS'u tutar.
- **string[] GetDirectories(string adres)**
- Belirtilen adresteki bütün klasörleri adresleriyle birlikte bir string dizisi olarak tutar.
- **string GetDirectoryRoot(string adres)**
- Belirtilen adresteki klasörün kök dizin bilgisini verir. Örneğin adres C:\Program Files\CONEXANT ise C:\ değerini döndürür.

Directory sınıfı

- **string[] GetFiles(string adres)**
- Belirtilen adresteki dosyaları adresleriyle birlikte string dizisi olarak tutar. Bu ve benzer metotlarda liste İngilizce alfabetik sırasına göredir. GetFiles() metodunun bir prototipi daha vardır:
- **string[] GetFiles(string adres, string dosya)**
- Adresteki dosya(lar) adresleriyle birlikte string dizisi olarak tutulur. Dosya isminde joker karakterleri (*, ?) kullanılabilir.
- **string[] GetFileSystemEntries(string adres)**
- Belirtilen adresteki bütün dosya ve klasörleri adresleriyle birlikte bir string dizisi olarak tutar.
- **DateTime GetLastAccessTime(string adres)**
- Belirtilen adresteki dosya ya da klasöre en son ne zaman erişildiğini DateTime türünden tutar.
- **DateTime GetLastWriteTime(string adres)**
- Belirtilen adresteki dosya ya da klasörün en son ne zaman değiştirildiğini DateTime türünden tutar.

Directory sınıfı

- **DateTime GetCreationTime(string adres)**
- Belirtilen adresteki dosya ya da klasörün ne zaman oluşturulduğunu DateTime türünden tutar.
- **string[] GetLogicalDrives()**
- Bilgisayardaki bütün sürücülerini bir string dizisi olarak tutar. Bu sürücülere her türlü sabit disk, CD-ROM sürücü, flash disk vb. dâhildir.
- **DirectoryInfo GetParent(string adres)**
- Belirtilen adresin bir üst klasörünü DirectoryInfo nesnesi olarak döndürür.
- **void Move(string kaynak_adres,string hedef_adres)**
- Dosya ve klasörleri bir konumdan başka bir konuma taşır. Örnekler:
 - Directory.Move(@"C:\Belgelerim\dosya.rtf", @"C:\Web\deneme.rtf");
 - Bu kod C:\Belgelerim altındaki dosya.rtf dosyasını C:\Web konumuna deneme.rtf adıyla taşır.
 - Directory.Move(@"C:\Web",@"C:\Belgelerim\internet");
 - Bu kod sürücü klasöründeki Web klasörünü tüm içeriğiyle birlikte C:\Belgelerim klasörüne internet adıyla taşır. Eğer hedef klasörde aynı adlı dosya ya da klasör varsa çalışma zamanı hatası alırsınız.

Directory sınıfı

- **void SetLastAccessTime(string adres, DateTime zaman)**
- Belirtilen adresteki dosya ya da klasörün en son erişim zamanını zaman olarak değiştirir.
- **void SetLastWriteTime(string adres, DateTime zaman)**
- Belirtilen adresteki dosya ya da klasörün en son değiştirilme zamanını zaman olarak değiştirir.
- **void SetCreationTime(string adres, DateTime zaman)**
- Belirtilen adresteki dosya ya da klasörün oluşturulma zamanını zaman olarak değiştirir.
- **void SetCurrentDirectory(string adres)**
- Programın çalıştığı klasörü belirtilen adres ile değiştirir.

Directory sınıfı

```
using System;
using System.IO;

class Program
{
    static void Main()
    {
        string yol = @"C:\Klasor";
        Directory.CreateDirectory(yol);
        Console.WriteLine("Klasör Oluşturuldu...");
        Console.ReadLine();
        Console.Clear();

        Console.WriteLine("Klasör Var mı: " + Directory.Exists(yol));
        Console.WriteLine("Oluşturulma Zamanı: " + Directory.GetCreationTime(yol));
        Console.WriteLine("Son Erişim Zamanı : " + Directory.GetLastAccessTime(yol));
        Console.WriteLine("Son Yazma Zamanı : " + Directory.GetLastWriteTime(yol));

        Console.WriteLine("Şu anki klasör: " + Directory.GetCurrentDirectory());
        Console.ReadLine();
        Console.Clear();

        Console.WriteLine("C Klasör Listesi:");
        string[] klasorler = Directory.GetDirectories(@"C:\");

        foreach (string s in klasorler)
            Console.WriteLine(s);

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

Directory sınıfı

```
Console.WriteLine("Kök Klasör:");  
Console.WriteLine(Directory.GetDirectoryRoot(yol));  
Console.ReadLine();  
Console.Clear();  
  
Console.WriteLine("C Dosya Listesi");  
string[] dosyalar = Directory.GetFiles(@"C:\");  
  
foreach (string s in dosyalar)  
    Console.WriteLine(s);  
  
Console.ReadLine();  
Console.Clear();  
  
Console.WriteLine("C Dosya ve Klasör Listesi");  
string[] tum = Directory.GetFileSystemEntries(@"C:\");  
  
foreach (string s in tum)  
    Console.WriteLine(s);  
  
Console.ReadLine();  
Console.Clear();  
  
Console.WriteLine("Mantıksal Sürücüler:");  
string[] mantiksalsuruculer = Directory.GetLogicalDrives();  
  
foreach (string s in mantiksalsuruculer)  
    Console.WriteLine(s);  
  
Console.ReadLine();  
Console.Clear();
```


Directory sınıfı

```
Console.WriteLine("Bir üst klasör:");  
Console.WriteLine(Directory.GetParent(@"C:\Windows\system32"));  
  
Console.ReadLine();  
Console.Clear();  
  
Directory.Move(@"C:\Klasor", @"C:\Windows\Klasor");  
Console.WriteLine("Klasör Taşındı...");  
Console.ReadLine();  
Console.Clear();  
  
Directory.SetCurrentDirectory(@"C:\Windows");  
  
Console.WriteLine("Klasör Silindi...");  
Directory.Delete("Klasor");  
}  
}
```


File sınıfı

- File sınıfındaki birçok metot Directory sınıfında da vardır, tek farkları aynı görevleri dosyalar için yerine getirmeleridir. Bu metotlar şunlardır: Exists(), Delete(), GetCreationTime(), GetLastAccessTime(), GetLastWriteTime(), Move(), SetCreationTime(), SetLastAccessTime(), SetLastWriteTime(). File sınıfının Directory sınıfında olmayan metotları:
- **StreamWriter AppendText(string adres)**
- Adreste belirtilen dosya için daha sonra göreceğimiz bir StreamWriter nesnesi oluşturur.
- **void Copy(string kaynak,string hedef)**
- Kaynakta belirtilen dosya hedefe kopyalanır. Kopyalamada bir isim çakışması söz konusuysa, yani sizin kopyalama yapmak istediğiniz klasörde zaten aynı isimli bir dosya varsa çalışma zamanı hatası alırsınız. Bunu önlemek içinse;
- **void Copy(string kaynak,string hedef,bool a)**
 - Burada a true olursa eğer hedef klasörde aynı adlı dosya varsa üstüne yazılır. a false yapılırsa iki parametrelili hâlden farkı kalmaz.

File sınıfı

- **FileStream Create(string adres,int tampon)**
- Belirtilen adresteki dosya oluşturulur ve dosyaya ilişkin FileStream nesnesi döndürülür. tampon yazılmazsa yani sadece bir parametre yazılırsa varsayılan tampon miktarı kullanılır.
- **StreamWriter CreateText(string adres)**
- Belirtilen adreste üzerine yazmak için bir text dosyası oluşturulur ve dosyaya ilişkin StreamWriter nesnesi döndürülür.
- **FileAttributes GetAttributes(string adres)**
- Belirtilen adresteki dosya ya da klasörün FileAttributes enumu cinsinden özniteliği döndürülür. FileAttributes enum sabiti şu sözcükleri içerir: Archive, Compressed, Device, Directory, Encrypted, Hidden, Normal, NotContentIndexed, Offline, ReadOnly, ReparsePoint, SparseFile, System, Temporary. Bir dosya ya da klasörün birden fazla özniteliği olabilir. Bu durumda öznitelikler virgülle ayrılır.

File sınıfı

- **Open**
- Üç farklı aşırı yüklenmiş çeşidi vardır. Bunlar:
 - FileStream Open(**string** adres, FileMode a)
 - FileStream Open(**string** adres, FileMode a, FileAccess b)
 - FileStream Open(**string** adres, FileMode a, FileAccess b, FileShare c)
 - Open() metodu belirtilen adresteki dosyayı açar ve dosyaya ilişkin FileStream nesnesini döndürür. FileMode, FileAccess ve FileShare System.IO isim alanında bulunan enumlardır ve dosyanın ne şekilde açılacağını ve üzerinde ne şekilde işlem yapılacağını belirtirler.
- **FileMode enumunda bulunan sözcükler**
 - **Append** Açılan dosyanın sonuna ekleme yapmak için kullanılır. Eğer dosya bulunmazsa oluşturulur.
 - **Create** Yeni dosya oluşturmak için kullanılır. Zaten dosya varsa üzerine yazılır.
 - **CreateNew** Yeni dosya oluşturmak için kullanılır, belirtilen dosya mevcutsa çalışma zamanı hatası verir.
 - **Open** Dosyayı açmak için kullanılır.
 - **OpenOrCreate** Belirtilen dosya varsa açılır, yoksa yenisi oluşturulur.
 - **Truncate** Belirtilen dosya açılır ve içeriği tamamen silinir.

File sınıfı

- **FileAccess enumunda bulunan sözcükler**
 - **Read** Dosya okumak için kullanılır.
 - **ReadWrite** Dosya okunmak ve yazılmak üzere açılır.
 - **Write** Dosya sadece yazılmak için açılır.
- **FileShare enumunda bulunan sözcükler**
 - **Inheritable** Dosyanın child (yavru) prosesler tarafından türetilebilmesini sağlar.
 - **None** Dosyanın aynı anda başka prosesler tarafından açılmasını engeller.
 - **Read** Dosyanın aynı anda başka proseslerce de açılabilmesini sağlar.
 - **ReadWrite** Dosyanın aynı anda başka proseslerce de açılıp, okunup, yazılabilmesini sağlar.
 - **Write** Dosyaya aynı anda başka proseslerce yazılabilmesini sağlar.

File sınıfı

- **FileStream OpenRead(string adres)**
- Belirtilen dosyayı yalnızca okumak için açar ve dosyaya ilişkin FileStream nesnesini döndürür.
- **StreamReader OpenText(string adres)**
- Belirtilen dosyayı yalnızca text modunda okumak için açar ve dosyaya ilişkin StreamReader nesnesini döndürür.
- **FileStream OpenWrite(string adres)**
- Belirtilen dosyayı yazma modunda açar ve dosyaya ilişkin FileStream nesnesini döndürür.

File sınıfı

```
using System;
using System.IO;

class Program
{
    static void Main()
    {
        string[] tumdosyalar = Directory.GetFileSystemEntries(@"C:\");

        foreach (string s in tumdosyalar)
        {
            Console.Write(s + " ==> ");
            Console.WriteLine(File.GetAttributes(s));
        }
    }
}
```

DirectoryInfo sınıfı

- DirectoryInfo sınıfı Directory sınıfının aksine static olmayan metot ve özellikleri içerir. Önce özellikleri bir örnek üzerinde görelim:
 - `using System;`
 - `using System.IO;`
 - `class DirectoryInfoSinifi {`
 - `static void Main() {`
 - `string adres= @"C:\WINDOWS" ;`
 - `DirectoryInfo d=new DirectoryInfo(adres);`
 - `Console.WriteLine("Özellikler: "+d.Attributes);`
 - `Console.WriteLine("Oluşturulma tarihi: "+d.CreationTime);`
 - `Console.WriteLine("Var mı? "+d.Exists);`
 - `Console.WriteLine("Uzantı: "+d.Extension);`
 - `Console.WriteLine("Tam adres: "+d.FullName);`
 - `Console.WriteLine("Son erişim zamanı: "+d.LastAccessTime);`
 - `Console.WriteLine("Son değişiklik zamanı: "+d.LastWriteTime);`
 - `Console.WriteLine("Klasör adı: "+d.Name); Console.WriteLine("Bir üst klasör: "+d.Parent);`
`Console.WriteLine("Kök dizin: "+d.Root); } }`

DirectoryInfo sınıfı

- DirectoryInfo sınıfının metotlarının tamamı static değildir. Bu metotların çalışması için gereken adres bilgisi, kendisine ulaşılması için kullanılan DirectoryInfo nesnesindedir.
- **void Create()**
- Klasör oluşturur.
- **DirectoryInfo CreateSubdirectory(string adres)**
- Belirtilen adreste bir alt dizin oluşturur.
 - Örneğin C:\deneme altında \deneme2\deneme3 dizini oluşturmak için;
 - **string** adres=@"C:\deneme";
 - DirectoryInfo d=new DirectoryInfo(adres);
 - d.Create();
 - DirectoryInfo alt=d.CreateSubdirectory("deneme2");
 - alt.CreateSubdirectory("deneme3");
 - CreateSubdirectory metodu kendisine ulaşılan nesne içinde parametredeki klasörü oluşturuyor ve oluşturduğu klasörü de DirectoryInfo nesnesi olarak döndürüyor.

DirectoryInfo sınıfı

- **Delete**
- İki farklı aşırı yüklenmiş versiyonu vardır.
 - `void Delete()`
 - `void Delete(bool a)`
 - Birincisinde klasör boşsa silinir, ikincisinde a true ise klasör, içindeki her şeyle silinir.
- **DirectoryInfo[] GetDirectories()**
 - İlgili klasörde bulunan bütün dizinleri bir DirectoryInfo dizisinde tutar.
- **FileInfo[] GetFiles()**
 - İlgili klasörde bulunan bütün dosyaları bir FileInfo dizisinde tutar.
- **FileSystemInfo[] GetFileSystemInfos()**
 - İlgili klasördeki bütün dosya ve klasörler bir FileSystemInfo dizisinde tutulur.
- **void MoveTo(string hedef)**
 - İlgili dizin, içindeki tüm dosya ve klasörlerle beraber hedefe taşınır.
- **void Refresh()**
 - İlgili klasörün özelliklerini diskten tekrar yükler.
- **NOT:** Bütün taşıma ve kopyalama işlemlerinde kaynak ve hedef aynı sürücüde olmalıdır.

DirectoryInfo sınıfı

```
using System;
using System.IO;

class Program
{
    static void Main()
    {
        string yol = @"C:\Windows\System32";

        DirectoryInfo d = new
            DirectoryInfo(yol);

        Console.Write("Özellikler:");
        Console.WriteLine(d.Attributes);

        Console.Write("Oluşturma Tarihi:");
        Console.WriteLine(d.CreationTime);

        Console.Write("Klasör var mı?");
        Console.WriteLine(d.Exists);

        Console.Write("Uzantı:");
        Console.WriteLine(d.Extension);
```

```
        Console.Write("Tam Yol:");
        Console.WriteLine(d.FullName);

        Console.Write("Son Erişim Zamanı:");
        Console.WriteLine(d.LastAccessTime);

        Console.Write("Son Yazma Zamanı:");
        Console.WriteLine(d.LastWriteTime);

        Console.Write("Klasör Adı:");
        Console.WriteLine(d.Name);

        Console.Write("Üst Klasör:");
        Console.WriteLine(d.Parent);

        Console.Write("Kök Klasör:");
        Console.WriteLine(d.Root);
    }
}
```

FileInfo sınıfı

- FileInfo özellikleri aşağıdaki örnek üzerinde gösterilmiştir.
 - **using** System;
 - **using** System.IO;
 - **class** FileInfoSinifi {
 - **static void** Main() {
 - **string** adres=@"C:\WINDOWS\deneme.txt";
 - FileInfo d=new FileInfo(adres);
 - Console.WriteLine("Öznitelikler: "+d.Attributes);
 - Console.WriteLine("Oluşturulma tarihi: "+
 - d.CreationTime); Console.WriteLine("Var mı? "+d.Exists);
 - Console.WriteLine("Uzantı: "+d.Extension);
 - Console.WriteLine("Tam adres: "+d.FullName);
 - Console.WriteLine("Son erişim zamanı: "+d.LastAccessTime);
 - Console.WriteLine("Son değişiklik zamanı: "+d.LastWriteTime);
 - Console.WriteLine("Boyut: "+d.Length);
 - Console.WriteLine("Klasör adı: "+d.Name);
 - Console.WriteLine("Bulunduğu klasör: "+d.DirectoryName); } }

FileInfo sınıfı

- FileInfo sınıfı File sınıfındaki AppendText(), Create(), CreateText(), Delete(), Open(), OpenRead(), OpenText() ve OpenWrite() metotlarını içerir. Bunlara daha önce değinilmişti.
- File sınıfında olmayan metotlarsa;
- **CopyTo**
- İki aşırı yüklenmiş versiyonu vardır:
 - FileInfo CopyTo (string hedef)
 - FileInfo CopyTo (string hedef, bool a)
 - Birincisinde ilgili dosya hedefe kopyalanır. Hedefte aynı adlı dosya varsa çalışma zamanı hatası alırsınız. İkincisinde a true ise, hedefte aynı adlı dosya varsa üzerine yazılır.
- **void MoveTo(string hedef)**
- İlgili dosya hedefe taşınır.
- **void Refresh()**
- İlgili dosyanın bilgileri diskten tekrar alınır.

FileInfo sınıfı

```
using System;
using System.IO;

class Program
{
    static void Main()
    {
        string yol = @"C:\Windows\clock.avi";

        FileInfo f = new FileInfo(yol);

        Console.Write("Özellikler:");
        Console.WriteLine(f.Attributes);

        Console.Write("Oluşturma Tarihi:");
        Console.WriteLine(f.CreationTime);

        Console.Write("Dosya var mı?");
        Console.WriteLine(f.Exists);

        Console.Write("Uzantı:");
        Console.WriteLine(f.Extension);
```

```
        Console.Write("Tam Ad:");
        Console.WriteLine(f.FullName);

        Console.Write("Son Erişim Zamanı:");
        Console.WriteLine(f.LastAccessTime);

        Console.Write("Son Yazma Zamanı:");
        Console.WriteLine(f.LastWriteTime);

        Console.Write("Boyut:");
        Console.WriteLine(f.Length);

        Console.Write("Dosya Adı:");
        Console.WriteLine(f.Name);

        Console.Write("Bulunduğu Klasör:");
        Console.WriteLine(f.DirectoryName);
    }
}
```

Path sınıfı

- Path sınıfı çeşitli işlemler yapan static üye elemanlara sahiptir. Örnek program:

- `using System; using System.IO;`
- `class PathSinifi {`
- `static void Main() {`
- `string adres=@"C:\dizin\deneme.txt";`
- `Console.WriteLine("Uzantı: "+Path.GetExtension(adres));`
- `string yeniAdres=Path.ChangeExtension(adres,"jpg");`
- `Console.WriteLine("Yeni uzantı: "+Path.GetExtension(yeniAdres));`
- `string adres2=@"C:\klasör";`
- `Console.WriteLine("Yeni adres: "+Path.Combine(adres,adres2));`
- `Console.WriteLine("Klasör: "+Path.GetDirectoryName(adres));`
- `Console.WriteLine("Dosya adı: "+Path.GetFileName(adres));`
- `Console.WriteLine("Uzantısız dosya adı: "+Path.GetFileNameWithoutExtension(adres));`
- `Console.WriteLine("Tam adres: "+Path.GetFullPath(adres));`
- `Console.WriteLine("Kök dizin: "+Path.GetPathRoot(adres));`
- `Console.WriteLine("Geçici dosya adı: "+Path.GetTempFileName());`

Path sınıfı

- `Console.WriteLine("Geçici dosya dizini: "+Path.GetTempPath());`
- `Console.WriteLine("Dosya uzantısı var mı? "+Path.HasExtension(adres));`
- `Console.WriteLine("Alt dizin ayırıcı: "+Path.AltDirectorySeparatorChar);`
- `Console.WriteLine("Dizin ayırıcı: "+Path.DirectorySeparatorChar);`
- `Console.Write("Geçersiz dosya adı karakterleri: ");`
- `char[] dizi=Path.GetInvalidFileNameChars();`
- `foreach(char b in dizi) Console.Write(b+" ");`
- `Console.Write("\nGeçersiz adres karakterleri: ");`
- `char[] dizi2=Path.GetInvalidPathChars();`
- `foreach(char b in dizi) Console.Write(b+" ");`
- `Console.WriteLine("\nAdres ayırıcı karakter: "+Path.PathSeparator);`
- `Console.WriteLine("Kök dizin ayırıcı: "+Path.VolumeSeparatorChar); }`
- `}`

Path sınıfı

- Uzantı: **.txt**
- Yeni uzantı: **.jpg**
- Yeni adres: **C:\klasör**
- Klasör: **C:\dizin**
- Dosya adı: **deneme.txt**
- Uzantısız dosya adı: **deneme**
- Tam adres: **C:\dizin\deneme.txt**
- Kök dizin: **C:**
- Geçici dosya adı: **C:\Users\ERKAN TANYILDIZI\AppData\Local\Temp\tmpA303.tmp**
- Geçici dosya dizini: **C:\Users\ERKAN TANYILDIZI\AppData\Local\Temp**
- Dosya uzantısı var mı? **True**
- Alt dizin ayırıcı: **/**
- Dizin ayırıcı: ****
- Geçersiz dosya adı karakterleri: " < > | ☺ ☹ ♥ ♦ ♣ ♠ 🎵 ⚙ ▶ ◀ ↕ !! ¶ § — † ↑ ↓ → ← ⇐ ↔ ▲ ▼ : * ? \ /
- Geçersiz adres karakterleri: " < > | ☺ ☹ ♥ ♦ ♣ ♠ 🎵 ⚙ ▶ ◀ ↕ !! ¶ § — † ↑ ↓ → ← ⇐ ↔ ▲ ▼ : * ? \ /
- Adres ayırıcı karakter: **;**
- Kök dizin ayırıcı: **:**

Path sınıfı

```
using System;
using System.IO;

class Program
{
    static void Main()
    {
        string yol = @"C:\windows\clock.avi";

        Console.Write("Uzantı: ");
        Console.WriteLine(Path.GetExtension(yol));

        Console.Write("Klasör: ");
        Console.WriteLine(Path.GetDirectoryName(yol));

        Console.Write("Dosya Adı: ");
        Console.WriteLine(Path.GetFileName(yol));

        Console.Write("Uzantısız Dosya Adı:");
        Console.WriteLine(Path.GetFileNameWithoutExtension(yol));

        Console.Write("Tam Yol: ");
        Console.WriteLine(Path.GetFullPath(yol));

        Console.Write("Kök Dizin: ");
        Console.WriteLine(Path.GetPathRoot(yol));
    }
}
```

Path sınıfı

```
Console.Write("Geçici Dosya Adı: ");  
Console.WriteLine(Path.GetTempFileName());  
  
Console.Write("Geçici Dosya Dizini: ");  
Console.WriteLine(Path.GetTempPath());  
  
Console.Write("Uzantı var mı? : ");  
Console.WriteLine(Path.HasExtension(yol));  
  
Console.Write("Alternatif Alt Dizin Ayracı: ");  
Console.WriteLine(Path.AltDirectorySeparatorChar);  
  
Console.Write("Dizin Ayracı: ");  
Console.WriteLine(Path.DirectorySeparatorChar);  
  
Console.Write("Geçersiz Karakterler: ");  
Console.WriteLine(Path.GetInvalidPathChars());  
  
Console.Write("Yol Ayracı: ");  
Console.WriteLine(Path.PathSeparator);  
  
Console.Write("Sürücü Ayracı: ");  
Console.WriteLine(Path.VolumeSeparatorChar);
```

```
}
```

```
}
```

Dosya yazma ve okuma işlemleri

- **FileStream sınıfı**
- FileStream sınıfı ile diskteki bir dosya açılır. StreamReader ve StreamWriter sınıflarıyla üzerinde işlem yapılır. Dosyalar üzerinde metin tabanlı ve bayt tabanlı işler yapılabiliriz. Bir FileStream nesnesi çok değişik yollarla oluşturulabilir.
- Örnekler:
 - `string adres=@"C:\Program Files\deneme.txt";`
 - `FileStream FSnesnesi1=new FileStream(adres,FileMode.OpenOrCreate);`
 - `FileStream FSnesnesi2=new FileStream(adres,FileMode.OpenOrCreate,FileAccess.Write);`
 - `FileStream FSnesnesi3=new FileStream(adres,FileMode.OpenOrCreate, FileAccess.Write, FileShare.None);`
 - `FileInfo Flnesnesi1=new FileInfo(adres);`
 - `FileStream FSnesnesi4=Flnesnesi1.OpenRead();`
 - `FileInfo Flnesnesi2=new FileInfo(adres);`

Dosya yazma ve okuma işlemleri

- `FileStream FSnesnesi5=Flnesnesi2.OpenWrite();`
 - `FileInfo Flnesnesi3=new FileInfo(adres);`
 - `FileStream FSnesnesi6=Flnesnesi3.Create();`
 - `FileInfo Flnesnesi4=new FileInfo(adres);`
 - `FileStream FSnesnesi7=Flnesnesi4.Open(FileMode.OpenOrCreate);`
-
- Dosyayla ilgili işlemimiz bittiğinde `FileStream` sınıfının `Close()` metodu ile `FileStream` nesnesi tarafından tutulan kaynaklar boşaltılır ve dosyayı başka prosesler kullanabilir hâle gelir. `Close()` metodunun kullanılışı:
 - `FSnesnesi1.Close();`

FileStream sınıfı ile yazma ve okuma

- FileStream sınıfının Read() ve ReadByte() metotları dosya akımından byte düzeyinde veri okumamızı sağlarlar.
- ReadByte() metodu akımdan okuma yapamadığı zaman geriye -1 değerini döndürür. ReadByte() metodunun prototipi:
- **int** ReadByte()
- Bu metot ile akımdan bir baytlık bilgi okunur ve akımdaki okuma pozisyonu bir artırılır ki tekrar okuma da aynı değer okunmasın. Okunan byte değeri inte dönüştürülüp int olarak tutulur.
- İkinci metodumuz ise:
- **int** Read(**byte**[] dizi, **int** baslangic, **int** adet)
- Bu metot ile adet kadar bayt akımdan okunur, okunan bu baytlar dizi dizisine baslangic indeksinden itibaren yerleştirilir. Geri dönüş değeri okunan byte sayısıdır.

FileStream sınıfı ile yazma ve okuma

- Şimdi komut satırı argümanı olarak adı alınan dosyanın içeriğini ekrana yazan bir program yazalım:
 - `using System; using System.IO;`
 - `class DosyaAkimi {`
 - `static void Main(string[] args) {`
 - `string adres=args[0]; FileStream fs=new FileStream(adres,FileMode.Open);`
 - `int OkunanByte;`
 - `while((OkunanByte=fs.ReadByte())!=-1) Console.Write((char)OkunanByte);`
 - `}`
- Bir dosya akımına bir byte yazmak için
- `void WriteByte(byte veri)`
- metodu kullanılır. Eğer yazma işlemi başarısız olursa çalışma zamanı hatası oluşur.
- Dosya akımına bir bayt dizisi yazdırmak içinse
- `void Write(byte[] dizi,int baslangic,int adet)`
- metodu kullanılır. Bu metot ile byte dizisinin baslangic indeksinden itibaren adet kadar elemanı akıma yazılır. Akımın konum göstericisi yazılan byte kadar ötelenir.

FileStream sınıfı ile yazma ve okuma

- Dosya akımına yazılan veriler dosya sistemindeki dosyaya hemen aktarılmaz. Dosya akımı tamponlama mekanizması ile çalıştığı için belirli bir miktar veri yazılana kadar dosya güncellenmez. Ancak FileStream sınıfının Flush() metodunu kullanarak istediğimiz anda tamponu boşaltıp dosyayı tampondaki bilgilerle güncelleyebiliriz.
- Örnek:
- `using System;`
- `using System.IO;`
- `class deneme { static void Main(string[] args)`
- `{ string dosya=args[0];`
- `FileStream d=new FileStream(dosya, FileMode.CreateNew, FileAccess.Write);`
- `Console.Write("Dosyanın içeriğini yazın: ");`
- `string icerik=Console.ReadLine();`
- `foreach(char a in icerik)`
- `d.WriteByte((byte)a);`
- `d.Flush(); } }`

FileStream sınıfı ile yazma ve okuma

- FileStream sınıfının önemli özellikleri:
- **bool CanRead** Akımdan okuma yapılıp yapılamayacağı öğrenilir.
- **bool CanSeek** Akımda konumlandırma yapılıp yapılamayacağı öğrenilir.
- **bool CanWrite** Akıma yazma yapılıp yapılamayacağı öğrenilir.
- **long Position** Akımdaki aktif konum bilgisi öğrenilir.
- **long Length** Akımın bayt olarak büyüklüğü öğrenilir.
- FileStream sınıfının önemli metotları:
- **void Lock(long pozisyon,long uzunluk)** Bu metotla akımın pozisyonundan itibaren uzunluk kadar alanı başka proseslerin erişimine kapatılır.
- **long Seek(long offset,SeekOrigin a)** Bu metotla akımın konumu SeekOrigin ile belirtilmiş konumdan offset byte kadar ötelenir. SeekOrigin enumunun içerdiği sözcükler şunlardır:
 - **Begin** Akımın başlangıç noktası
 - **Current** Akımın o anda bulunduğu nokta
 - **End** Akımın en son noktası

FileStream Sınıfı

```
using System;
using System.IO;

class Program
{
    static void Main(string[] args)
    {
        string yol = @"Dosya.txt";
        FileStream fsw = new FileStream(yol, FileMode.Create, FileAccess.Write);

        byte[] veri = new byte[20];

        for (int i = 0; i < veri.Length; i++) veri[i] = (byte)(i + 1);

        fsw.Write(veri, 0, veri.Length);
        Console.WriteLine("{0} byte yazıldı...", veri.Length);

        fsw.Close();

        FileStream fsr = new FileStream(yol, FileMode.Open, FileAccess.Read);

        byte[] okunan = new byte[fsr.Length];

        fsr.Read(okunan, 0, (int)fsr.Length);

        foreach (byte b in okunan) Console.Write(b + " ");

        fsr.Close();
    }
}
```

StreamReader sınıfı

- StreamReader sınıfı FileStream sınıfının aksine baytlarla değil, metinlerle ilgilenir. Bir StreamReader nesnesinin oluşturulma yöntemleri:
 - `string dosya=@"C:\deneme\ornek.txt";`
 - `FileStream fs=new FileStream(dosya,FileMode.Open);`
 - `StreamReader sr1=new StreamReader(fs);`
 - `StreamReader sr2=new StreamReader(dosya);`
 - `FileInfo fi=new FileInfo(dosya);`
 - `StreamReader sr3=new StreamReader(fi);`
- Diğer sınıflarda olduğu gibi StreamReader nesneleriyle işimiz bittiğinde Close() metodunu kullanarak kaynakların iade edilmesi tavsiye edilir.
- StreamReader sınıfının önemli metotları:
- `string ReadLine()`
- Akımdan bir satırlık veriyi okur ve string olarak tutar. Eğer veri okunamazsa null değeri tutar. Okunan satırın sonuna "\n" eklenmez.

StreamReader sınıfı

- **string** ReadToEnd()
- Akımdaki verilerin tamamını string olarak tutar. Okuma işlemi aktif konumdan başlayacaktır. Eğer okuma yapılamazsa boşluk tutar.
- **int** Read()
- Akımdan bir karakterlik bilgi okunur ve int'e dönüştürülür. İşlem başarısız olursa -1 ile geri döner.
- **int** Read(**char**[] dizi,**int** indeks,**int** adet)
- Akımdan adet kadar karakteri dizi[indeks] elemanından itibaren diziye yerleştirir. Yerleştirilen karakter sayısını döndürür.
- **int** Peek()
- Akımdan bir karakterlik bilgi okunur ve bu karakterin inte dönüşmüş hâli ile geri dönülür. İşlem başarısız olursa -1 ile geri döner. En önemli nokta ise konum göstericisinin yerinin değiştirilmemesidir.

StreamReader sınıfı

- Örnek:
- `using System;`
- `using System.IO;`
- `class Deneme {`
- `static void Main() {`
- `string dosya="deneme.txt";`
- `FileStream fs=new FileStream(dosya,FileMode.Open);`
- `StreamReader sr=new StreamReader(fs);`
- `string satir;`
- `while((satir=sr.ReadLine())!=null)`
- `Console.WriteLine(satir);`
- `fs.Close();`
- `}`
- `}`

StreamWriter sınıfı

- StreamReader sınıfı ile dosyalardan text tabanlı veriler okunabiliyordu. StreamWriter sınıfı ise bunun tam tersini yapar. Yani StreamWriter sınıfı ile dosyalara text tabanlı veriler yazılır. Bir StreamWriter nesnesi şu yollarla oluşturulabilir:
 - `string dosya=@"C:\dosya.txt";`
 - `FileStream fs=new FileStream(dosya,FileMode.Open);`
 - `StreamWriter sw1=new StreamWriter(fs);`
 - `StreamWriter sw2=new StreamWriter(dosya);`
 - `FileInfo fi=new FileInfo(dosya);`
 - `StreamWriter sw3=fi.CreateText();`
- StreamReader sınıfında olduğu gibi StreamWriter sınıfında da Close() metoduyla StreamWriter nesnesine ilişkin kaynaklar iade edilir.

StreamWriter sınıfı

- StreamWriter sınıfının en önemli metotları:
- **void Write(string str)**
- Bu metotla akıma str yazısı eklenir. Yazının sonuna herhangi bir sonlandırıcı karakter konmaz. Bu metot ile diğer bütün veri türlerinden veri eklemek mümkündür. Örneğin: Write(5), Write(true), Write('c')
- **void WriteLine(string str)**
- Write metoduyla aynı işi yapar. Tek fark eklenen yazının sonuna kendisi "\n" stringini koyar. Ayrıca Write metodundan farklı olarak WriteLine() metodunu parametresiz kullanabiliriz. Bu durumda akıma sadece "\n" eklenir.
- **void Flush()**
- Tampondaki bilgilerin boşaltılmasını ve dosyanın güncellenmesini sağlar.
- Ayrıca StreamWriter sınıfının NewLine özelliği ile satır ayırıcı olan karakterleri belirleyebiliriz. Varsayılan olarak bu karakter "\n" ve "\r"dir.

StreamWriter sınıfı

- Örnek:
- `using System;`
- `using System.IO;`
- `class Deneme {`
- `static void Main(string[] args) {`
- `string dosya=args[0];`
- `FileStream fs=new FileStream(dosya,FileMode.Append,FileAccess.Write);`
- `StreamWriter sw=new StreamWriter(fs);`
- `Console.Write("Yazınızı girin: ");`
- `string yazi=Console.ReadLine();`
- `sw.Write(yazi);`
- `sw.Flush();`
- `}`
- `}`

StreamWriter Sınıfı

```
using System;
using System.IO;

class Program
{
    static void Main(string[] args)
    {
        string yol = @"C:\Dosya.txt";

        FileStream fs = new FileStream(yol, FileMode.Append, FileAccess.Write);
        StreamWriter sw = new StreamWriter(fs);

        while (true)
        {
            string giris = Console.ReadLine();
            if (giris.ToLower() == "x") break;
            sw.WriteLine(giris);
        }
        sw.Flush(); sw.Close();

        Console.Clear();

        StreamReader sr = new StreamReader(yol);

        string satir; int i=1;

        while ((satir = sr.ReadLine()) != null)
        {
            Console.WriteLine("{0}:{1}", i, satir);
            i++;
        }
        sr.Close(); fs.Close();
    }
}
```


BinaryWriter ve BinaryReader sınıfları

- BinaryWriter ve BinaryReader sınıflarının yaptığı iş StreamReader ve StreamWriter sınıflarının yaptığı işin aynısıdır.
- Ancak BinaryWriter ve BinaryReader sınıflarıyla StreamReader ve StreamWriter sınıflarından farklı olarak her türde veriyi akıma yazdırabiliriz, yani verinin illaki string olma zorunluluğu yoktur.
- Bu sınıflarda sırasıyla Write(int), Write(char), Write(char[]), Write(byte) ve ReadByte(), ReadChar(), ReadUInt32, ReadString() vb. metotlar bulunmaktadır.
- Bu metotlar bütün temel veri türleri için bildirilmiştir.

BinaryWriter ve BinaryReader sınıfları

- `using System;`
- `using System.IO;`
 - `class Deneme {`
 - `static void Main()`
 - `{ int i=5; decimal d=15.54M; char c='A'; string dosya="deneme.txt";`
 - `FileStream fs1=new FileStream(dosya,FileMode.OpenOrCreate);`
 - `BinaryWriter bw=new BinaryWriter(fs1); bw.Write(i); bw.Write(d);`
 - `bw.Write(c); bw.Close();`
 - `FileStream fs2=new FileStream(dosya,FileMode.Open);`
 - `BinaryReader br=new BinaryReader(fs2); Console.WriteLine(br.ReadInt32());`
 - `Console.WriteLine(br.ReadDecimal());`
 - `Console.WriteLine(br.ReadChar()); br.Close();`
- `}}`
- **NOT:** BinaryWriter sınıfıyla oluşturulan bir dosyayı Notepad ile okumaya kalkarsanız anlamsız simgelerle karşılaşacaksınız. Çünkü BinaryWriter sınıfı dosyalara text yöntemiyle değil, binary yöntemle kayıt yapar. BinaryReader sınıfı da bu binary kayıtları okur.

Console I/O işlemleri

- I/O işlemleri için gerekli olan sınıflardan System.IO isim alanında olmayan tek sınıf Console sınıfıdır. Console sınıfını ekrana bir şeyler yazmak için ya da kullanıcıdan girdi almak için sıklıkla kullanıldı.
- Konsol I/O işlemleri için önceden tanımlanmış üç tane standart akım mevcuttur. Bu akımlar **TextWriter** türü olan **Console.Out**, **Console.Error** ve **TextReader** türünden olan **Console.In** 'dir. Konsol ekranına yazdığımız yazılar aslında TextWriter sınıfının metotları ile olmaktadır. Console.WriteLine ise bizim için sadece bir aracılık görevi yapar. Örneğin aşağıdaki her iki satır da eş görevdedir. İkisi de ekrana merhaba yazar.
- `Console.Out.WriteLine("merhaba"); Console.WriteLine("merhaba");`
- Yani özetle Out, Console sınıfına bağlı bir özelliktir ve geri dönüş tipi TextWriter'dır ve bu veriyle TextWriter sınıfının static olmayan bir metodu olan WriteLine()'a erişiriz.
- Konsol ekranına yazı yazdırmak için Console sınıfının WriteLine() ve Write() metotlarını kullanırız. Bu ikisi arasındaki tek fark WriteLine'ın yazının sonuna "\n" ekleyip, Write'ın eklememesidir.

Console I/O işlemleri

- Konsoldan bilgi almak için Read() ve ReadLine() metotlarını kullanırız. Eğer tamponda herhangi bir veri yoksa Read() metodu kullanıcıdan veri girişi ister ve girilen stringteki ilk karakteri int olarak tutar. Sonraki Read() metotları ise o stringteki diğer karakterleri tutar. Eğer veri okunamazsa -1 değeri tutar.
- Örnek:
- `Console.Write((char)Console.Read()+" "+(char)Console.Read()+" "+(char)Console.Read());`
- Bu satırda önce kullanıcıdan veri girişi istenir. Diyelim ki kullanıcı **deneme** girmiş olsun. Bu durumda ekrana
- **d e n**
- yazılır. ReadLine() metodu Read() metodunun aksine ekrandan girdileri string olarak alır. Console.In özelliğini kullanarak da Read() ve ReadLine() metotlarını çağırabiliriz. Ayrıca Console.Out özelliğini kullanarak da Write() ve WriteLine() metotlarını kullanabiliriz.

Console I/O işlemleri

- Console.In'den erişebileceğimiz yani TextReader sınıfının diğer metotları:
- **int** Peek()
 - Bu metot ile standart girdi akımından bir karakter okunur ancak bu karakter tampondan silinmez.
 - Örnek: Console.Write((char)Console.In.Peek()+" "+(char)Console.In.Peek()+" "+(char)Console.In.Peek());
 - Burada kullanıcının ekrana **deneme** yazdığını varsayarsak ekrana **d d d** yazılacaktır.
- **int** ReadBlock(char[] dizi,int indeks,int adet)
 - Bu metot ile standart girdi akımından adet kadar karakter diziye indeks elemanından itibaren yerleştirilir.
- **string** ReadToEnd()
 - Bu metot ile standart girdi akımındaki bütün veriler okunarak tampondan temizlenir. Okunan veriler string nesnesi olarak döndürülür.

Standart akımların yönlendirilmesi

- C#'ta bütün I/O işlemleri akımlar (stream) üzerine kuruludur. Standart akımları başka akımlara yönlendirmek mümkündür. Örnek olarak komut satırında bir programı şöyle çalıştırırsak
- `programadi > deneme.txt`
- Bu programın, ekran çıktılarını `deneme.txt` dosyasına yazacağı anlamına gelir. Bu da standart çıktı olan konsolun başka akıma yönlendirilmesi anlamına gelir. Komut satırından
- `programadi < deneme.txt`
- Burada da `Console.ReadLine()` ya da `Console.Read()` yerine `deneme.txt` dosyasındaki metin kullanılır. C# programlarımız içinde akımları yönlendirmek içinse `Console` sınıfının şu metotları kullanılır.
- `static void SetOut(StreamWriter sw) veya static void SetOut(TextWriter tw)`
`static void SetError(StreamWriter sw) veya static void SetError(TextWriter tw)`
`static void SetIn(StreamReader sr) veya static void SetIn(TextReader sr)`

Standart akımların yönlendirilmesi

- Örnek: Console.In akımını SetIn metodu ile bir dosya akımına yönlendirerek girişi klavyeden değil, dosyadan alsın;
- `using System;`
- `using System.IO;`
- `class Deneme {`
- `static void Main() {`
- `FileStream fs=new FileStream("deneme.txt",FileMode.Open);`
- `Console.SetIn(new StreamReader(fs));`
- `Console.WriteLine(Console.ReadLine());`
- `Console.WriteLine(Console.ReadLine()); }`
- `}`
- `// burada ReadLine ile kullanıcıdan giriş alınmaz, dosyadan alır`

Standart akımların yönlendirilmesi

- Örnek: Console.WriteLine() metodu da dosyaya yönlendirilebilir. Tabii ki bunun için Console.SetOut metodu kullanılmalıdır. Programın içindeki hata mesajlarını ise SetError metodu ile bir dosyaya yazdırılabilir.
 - `using System;`
 - `using System.IO;`
 - `class Deneme {`
 - `static void Main() {`
 - `Console.WriteLine("Bu bir denemedir.");`
 - `FileStream fs1 = new FileStream("deneme.txt", FileMode.Create);`
 - `TextWriter tw = Console.Out;`
 - `StreamWriter sw = new StreamWriter(fs1);`
 - `Console.SetOut(sw); Console.WriteLine("Dosyaya yazma yapıyoruz."); //dosyaya`
 - `Console.SetOut(tw); Console.WriteLine("Merhaba dünya"); //ekrana`
 - `sw.Close(); } }`
- Burada önce ekrana Bu bir denemedir. yazılıyor. Sonra yeni bir FileStream nesnesi oluşturuluyor. Sonra hedefi konsol ekranı olan bir TextWriter nesnesi oluşturuluyor. Sonra ilişiği deneme.txt olan bir StreamWriter nesnesi oluşturuluyor. Sonra çıktı birimi deneme.txt olarak değiştiriliyor. Sonra ekrana -daha doğrusu dosyaya- Dosyaya yazma yapıyoruz. yazdırılıyor. Sonra çıktı birimi yeniden ekran yapılıyor. Sonra ekrana Merhaba dünya yazdırılıyor. En son da sw akımının kaynakları boşaltılıyor.

Visual Studio.Net -C#

9. HAFTA

FORM UYGULAMALARI