

1.2 .NET ve Java Platformları

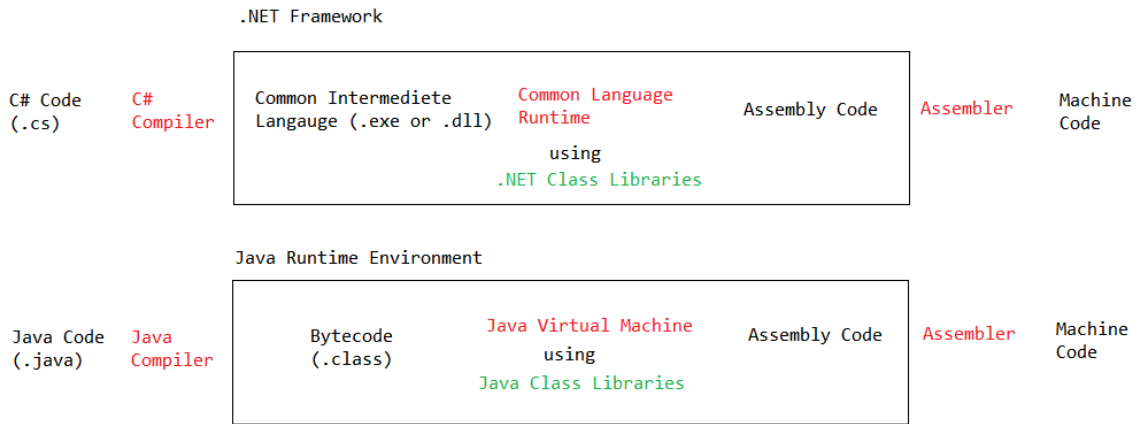
.NET Framework, Microsoft tarafından geliştirilen, açık İnternet protokolleri ve standartları üzerine kurulmuş bir "uygulama" geliştirme platformudur.

Buradaki uygulama kavramının kapsamı çok geniştir. Bir masaüstü uygulamasından bir web tarayıcı uygulamasına kadar her şey bu platform içinde düşünülmüştür ve desteklenmiştir.

Bu platform, işletim sisteminden ve donanımdan daha üst seviyede taşınabilir olarak tasarlanmıştır.

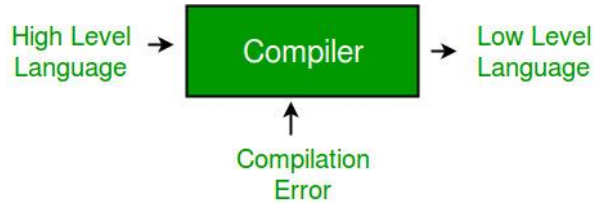
.NET framework Windows ortamına yönelik uygulama geliştirme mimarisi iken, .NET Core çapraz platform desteği sunun uygulama geliştirme mimarisidir ve açık kaynak kodludur. .NET Framework 4.8 ve .NET Core 6.0.7 versiyonuna sahiptir.

.NET'e alternatif olarak geliştiricilerin diğer bir tercihi JAVA platformlarında Java dilini kullanarak uygulama geliştirmektir.



Şekil 2-1. C# ve Java derleme yolları (stackoverflow.com).

1.2.1 Compile Time ve Run Time



Compile Time (Derleme Zamanı)

Compile işlemi; “yüksek seviyeli bir dilin daha düşük seviyeli bir dile dönüştürülmesi işlemidir”. Yazılmış kod, compile işleminden sonra makinelerin seviyesine yaklaşmaktadır. İşte bu işlemi gerçekleştiren mekanizmaya Compiler(Derleyici), bu süreçte yaşanan tüm işlemleri zaman açısından anlatmak için de Compile Time (Derleme Zamanı) kavramları kullanılır.

Yukarıdaki compiler diyagramında Compilation Error kısmı ise, yüksek seviyeli bir dilin compile edilme işlemi sırasında oluşan hataları anlatmaktadır. Compile Time hataları exe dosyası daha oluşturulmadan derleme işlemi sırasında görülür. (Örnek ‘Syntax Error - Sözdizimi Hatası’).

Run Time (Çalışma Zamanı)

Programın çalıştırıldığı andan sonlandırılincaya kadar geçen süreci anlatmak için kullanılan kavramdır. Run Time’da alınan hataları bulmak için çok fazla efor sarfetmek gerekebilir. Mesela: Bellek yetersiz hatası, dosya bulunamadı hatası vs.

1.2.2 .Net Derleme

.Net ortamında derleme, C ve Pascal dillerinde ki gibi direkt makine koduna derlenmez. Önce IL dediğimiz bir ara koda derlenir. (Javada da olduğu gibi)

Bu derlenen ilk kodun dosyasına assembly denir ve uzantısı exe'dir.

Bu dosya çalıştırılmak istendiğinde ise .Net Framework devreye girer ve IL kodu bilgisayarın anlayabileceği makine koduna dönüştürür (alttaki şekil). İşte bu yüzden de yazdığımız programın bir bilgisayarda çalışması için o bilgisayarda .Net Framework programının kurulu olması gerekir, çünkü .Net Framework IL kodu bilgisayarın anlayabileceği koda çevirir.



.Net Framework, oluşturduğu makine kodlarını geçici bir süreliğine belleğe koyar, eğer aynı kodlar tekrar çalıştırılmak istenirse tekrar IL koddan makine koduna dönüşüm yapmak yerine bu belleğe kaydettiği makine kodlarını kullanır. Bu yüzden oluşturduğumuz programımızı ilk çalıştırdığımız zaman programımız biraz yavaş çalışabilir, ancak daha sonraki çalışmalarda hızlanacaktır.

1.2.3 IL kodu (Ara Kod)

CIL (Common Intermediate Language) veya .NET Framework için MSIL olarak adlandırılır. Türkçe'ye Ortak Ara Dil olarak çevrilmektedir ve C# kadar yüksek seviyeli yani bir insanın rahatça anlayabileceği kadar kolay bir dil olmamasına karşın, makine dili kadar da insandan uzak bir dil değildir.

Kodun direkt makine kodu yerine, önce IL koda çevrilmesinin bazı avantajları vardır. Bunlardan en önemlisi programımızın farklı işletim sistemlerinde çalışmasının eskiye oranla çok daha kolay olmasıdır. Çünkü makine kodu taşınabilir değildir, programları direkt makine koduna derlediğimiz zaman ilgili programın belirli bir işletim sistemine göre derlenmesi gerekir. Fakat IL kod taşınabilir, ortak bir koddur, işletim sistemlerindeki çeşitli programlar vasıtasıyla makine koduna dönüştürülebilir. Örneğin Windows'ta bu işi .Net Framework yaparken, Linux'ta Mono yapabilir.

Derleme işlemi sonrası CIL'e dönüştürülen kodlarımız exe dosyası olarak kaydedilir. Burada .exe içerisinde direkt olarak makine kodları değil, yazdığımız koddan CIL'e dönüştürülmüş kodlar yer almaktadır. Bu sebeple exe dosyasının çalışması için makinede muhakkak .NET Framework'ün bulunması gerekmektedir. Çünkü çalıştırma işlemini .NET Framework içerisindeki CLR mekanizması gerçekleştirecektir.

CIL içerisinde Metadata (üst veri) olarak adlandırılan bir yapı daha vardır. Metadata programda kullanılan veri tiplerini, oluşturduğumuz sınıfları, sınıfların methodlarını, özelliklerini ve diğer bilgilerini de içerir.

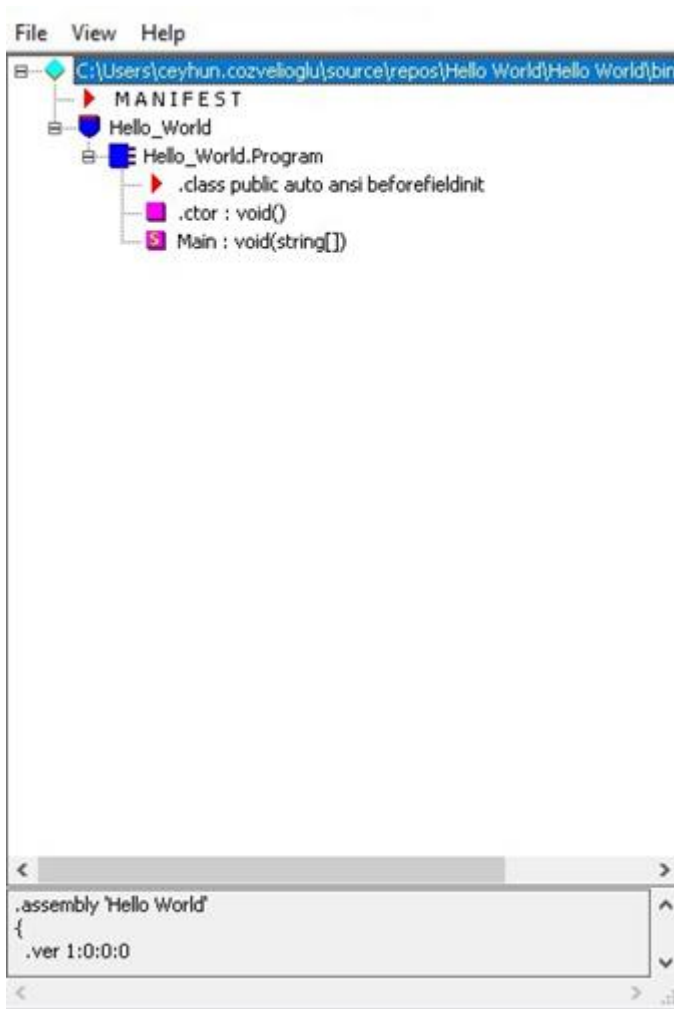
Örnek: Visual Studio ile basit bir Hello World console application projesi oluşturalım:

```

1  using System;
2
3  namespace Hello_World
4  {
5      public class Program
6      {
7          static void Main(string[] args)
8          {
9              Console.WriteLine("Hello World");
10         }
11     }
12 }

```

Kodu derledikten sonra, CIL kodunu görüntüleyebilmek için ILDASM tool'unu kullanabiliriz. Bu tool'a Developer Command Prompt'a ildasm yazarak ulaşıp exe dosyasını açtığımızda:



```

Hello_World.Program::Main : void(string[])
Find Find Next
.method private hidebysig static void Main(string[] args) cil managed
{
    .entrypoint
    // Code size      13 (0xd)
    .maxstack 8
    IL_0000: nop
    IL_0001: ldstr      "Hello World"
    IL_0006: call       void [mscorlib]System.Console::WriteLine(string)
    IL_000b: nop
    IL_000c: ret
} // end of method Program::Main

```

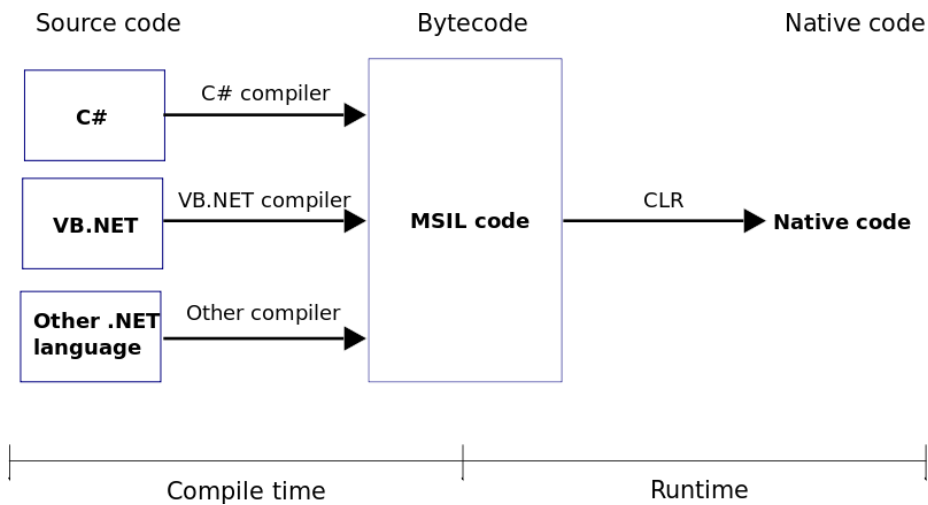
Şeklinde programın içeriğine ulaşırız.

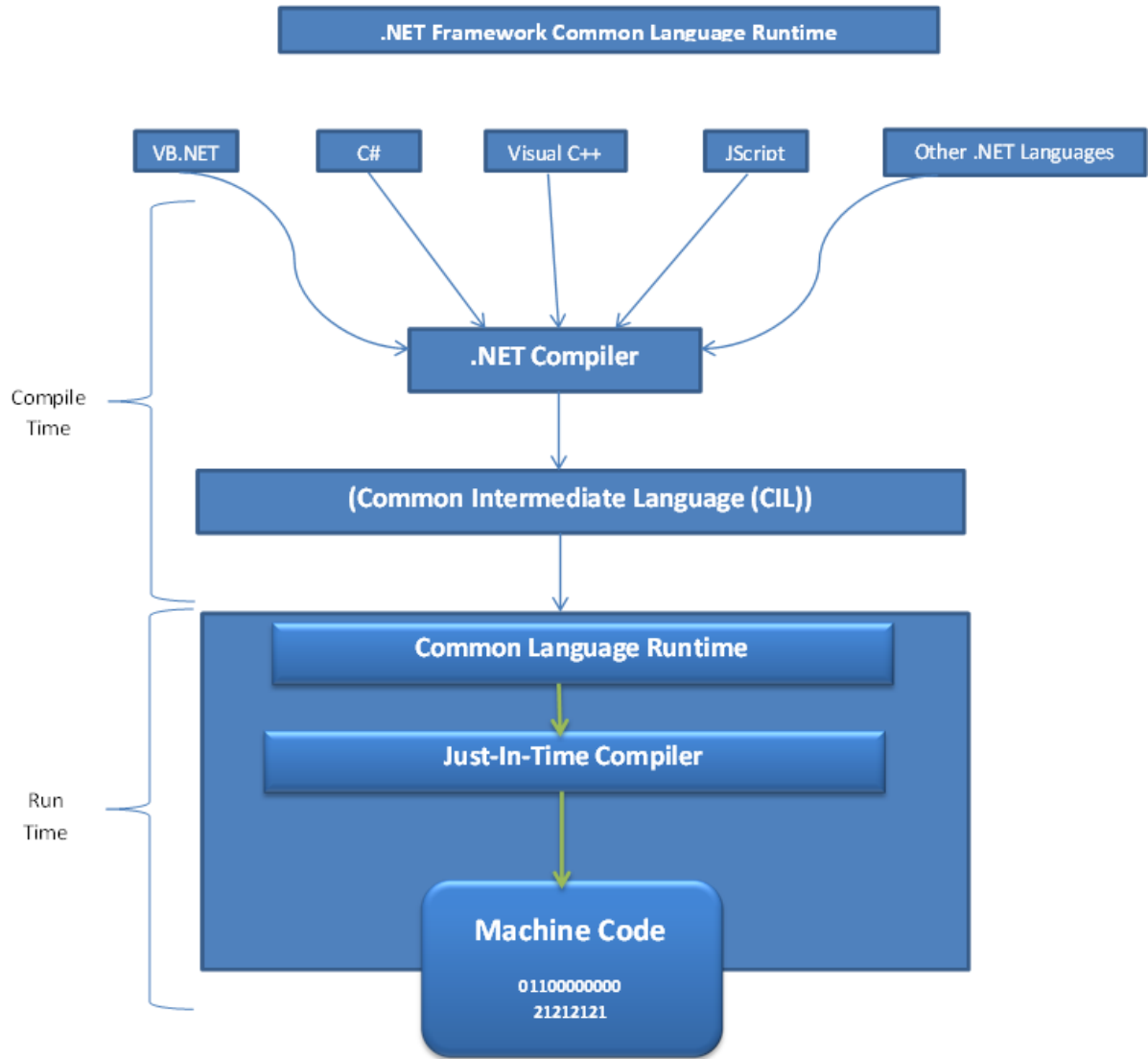
1.2.4 Common Language Runtime (CLR - Ortak Dil Çalışma Zamanı)

CLR (Common Language Runtime) .NET Framework'te programların çalışmasını kontrol eden ve işletim sistemi ile program arasında yer alan bir arabirimdir. CIL kodlarını çalıştıran ana mekanizmadır. CIL kodları daha önce de incelediğimiz gibi makine dili değil bir ara dildir. Bu sebeple programın çalıştırılabilmesi için bu ara dili alıp makine diline çevirecek ve makinede işletecek bir mekanizmaya ihtiyaç duymaktaydık. Bu ihtiyacı CLR karşılamaktadır.

CLR, içerisinde yer alan JIT (Just in Time) derleyicisi ile CIL kodunu alır makine koduna çevirir ve programın çalışmasını sağlar. CLR'ın bize sağladığı avantaj programlarımızı platform bağımsız çalıştırabilmemizi sağlamasından ileri geliyor. .NET Framework yüklü herhangi bir makinede CIL kodunu CLR işleteceği için kaynak kullanımları, performans ile ilgili maksimum optimizasyonu da yine JIT derleyicisi ile beraber sağlamış oluyor.

CLR şu şekilde çalışır;

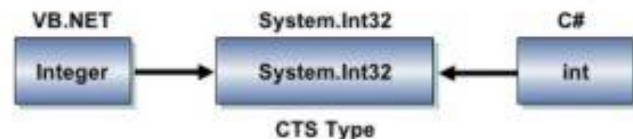




Şekil 2-2. The process of creating runnable software in Microsoft's .NET framework

1.2.5 CTS (Common Type System)

Bütün veri tiplerinin tanımlı olduğu bir sistem olarak düşünebiliriz. CTS sayesinde .NET Framework'te farklı dillerde yazılmış programların veri tipleri arasındaki uyum sağlanmış olur.



Örneğin VB.NET ile yazılmış bir Integer değişkeni, C#'ta int olarak yazılsa da CTS karşılıkları System.Int32'dir ve RAM'de kapladıkları alan aynıdır. Bu sayede tipler arasındaki uyum sağlanmış olur, değişen tek şey dillerdeki syntax olmaktadır.

1.2.6 Makine Dili

Makine dili, işlemcinin verilen komutlar doğrultusunda çalıştırılmasını sağlayan ve işlemci mimarisine göre değişen en alt seviyedeki programlama dilidir.

Bu dil sadece 0 ve 1 binary ikililerinin anlamlı kombinasyonlarından meydana gelmektedir. Bu nedenle, makine dilinin anlaşılması güçtür.

0 ve 1 ikilileri işlemcinin instruction seti doğrultusunda işleme (process) uygulanacak operasyon, operasyonun gerçekleştirileceği verinin hafızada bulunduğu adres ve hafızaya ulaşım yolları gibi bilgileri ifade edecek şekilde bir araya gelmekte ve işlemci tarafından decode edilerek gerekli işlemin yerine getirilmesi sağlanmaktadır.

Diğer programlama dillerin gerektirdiği derleyici ya da yorumlayıcı kullanımını gerektirmediğinden ve donanımı doğrudan kontrol etme gücü olduğundan kullanılır.

1.2.7 Assembly

Assembly Dili (Assembly Language) bilgisayar programlarının yazılmasında kullanılan alt seviyeli bir dildir. Assembly dili programlarının yazılımında insan dostu sembollerin “mnemonics” kullanılması, daha fazla hataya yatkın ve zaman alıcı ilk bilgisayarlarda kullanılmış olan bir hedef bilgisayarının sayısal makine kodunda doğrudan programlama çalışmasının yerine geçmiştir.

Bir assembly dil programı, “Assembler” olarak adlandırılan faydalı bir program tarafından hedef bilgisayarın makine koduna çevrilir. (Bir çevirici bir derleyiciden (compiler) farklıdır, bu genellikle “mnemonic” ifadelerden makine komutlarına teke tek (izomorfik) çeviriler yapar.)

Assembly dili karmaşık programlar yazmak için kullanılan düşük seviyeli bir programlama dilidir. Assembly insanlar tarafından anlaşılması zor olan makina dilinin sayısal ifadelerini, insanlar tarafından anlaşılabilir programlanması daha kolay olan alfabetik ifadelerle değiştirerek düşük seviyede programlama için bir ortam oluşturur.

Assembly kullanmanın amacı, ilk bilgisayarlarda yazılan programların daha az hata içermesi ve daha az zaman almasını sağlamaktır.

```

title    Hello World Program

dosseg
.model small
.stack 100h

.data
hello_message db 'Hello World!',0dh,0ah,'$'

.code
main proc
    mov     ax,@data
    mov     ds,ax

    mov     ah,9
    mov     dx,offset hello_message
    int     21h

    mov     ax,4C00h
    int     21h
main endp
end main

```

```

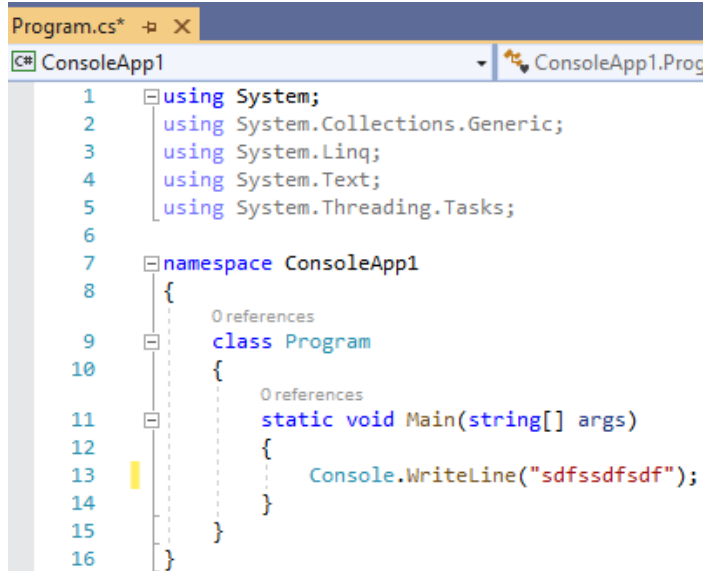
#include<stdio.h>

int main()
{
    printf("Hello World\n");
    return 0;
}

```


2 C# Temelleri

2.1 Konsol (Console) Uygulaması



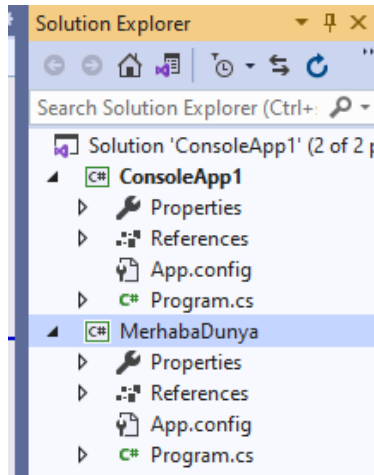
```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace ConsoleApp1
8 {
9     class Program
10    {
11        static void Main(string[] args)
12        {
13            Console.WriteLine("sdfssdfsdf");
14        }
15    }
16 }
```

2.2 Using

Using; isim uzayları (namespace) içerisindeki sınıfların kod içerisinde kullanılabilmesini sağlar.

2.3 Solution Explorer

Solution Explorer’da tüm projeler görülür. Yeni bir proje eklenebilir. (Add- >New Project). “Set as Startup Project” yada “Set Startup Projects” komutları ile açılışta çalışacak proje belirlenir.



2.4 Değişken Tanımlama

<DeğişkenTürü> <DeğişkenAdı>;

<DeğişkenAdı> = <DeğişkenDeğeri>;

//veya

<DeğişkenTürü> <DeğişkenAdı> = <DeğişkenDeğeri>;

2.4.1 Değişken Tanımlama Örnekleri

- `int sayi = 5;`
- `int x, y = 8, z; //Aynı türdeki değişkenler aynı anda tanımlanabilir.`
- `bool aktif = true;`
- `float f = 5.4f; // Değerin sonuna eklediğimiz f harfi değişkenin float türünde olduğunu gösterir.`
- `double d = 3.2;`
- `long l = 123456789;`
- `short s = -312;`
- `decimal dec = -5.26m; //Değerin sonundaki m harfi değişkenin decimal türünde olduğunu gösterir.`
- `char ch = 'c'; //Char tipinde ki değişkenler tek tırnak içerisinde yazılır.`
- `string deger = "merhaba";`

2.5 Veri Türleri Arasında Dönüşüm

2.5.1 Bilinçsiz Dönüşüm (implicit type conversion)

C#'ta düşük kapasiteli bir değişken, sabit ya da değişken ve sabitlerden oluşan matematiksel ifade daha yüksek kapasiteli bir değişkene atanabilir. Buna bilinçsiz tür dönüşümü denir, bunun için herhangi bir özel kod gerekmez.

```
static void Main(string[] args)
{
    Console.WriteLine("İlk Program");
    byte a = 5;
    short b = 10;
    sbyte c = 30;
    int d = a + b + c;
    string e = "deneme";
    char f = 'k';
    object g = e + f + d;
    long h = d;
    float i = h;
    double j = i;
    double k = 12.5f;
    Console.WriteLine(j + k);
    Console.ReadKey();
}
```

Bilinçsiz tür dönüşümüyle ilgili ilginç bir durum söz konusudur. `char` türünü kendisinden daha kapasiteli bir sayısal türe bilinçsiz olarak dönüştürebiliriz. Bu durumda ilgili karakterin `Unicode` karşılığı ilgili sayısal değişkene atanacaktır. Aşağıdaki örnekte "65".

```
static void Main(string[] args)
{
    Console.WriteLine("İlk Program");
    char a = 'A';
    int b = a;
    Console.WriteLine(b);
    Console.ReadKey();
}
```

2.5.2 Bilinçli Dönüşüm (explicit type conversion)

```
static void Main(string[] args)
{
    Console.WriteLine("İlk Program");
    int a = 5;
    byte b = (byte)a;
    Console.WriteLine(b);
    Console.ReadKey();
}
```

```
static void Main(string[] args)
{
    Console.WriteLine("İlk Program");
    byte b = (byte)27.5f;
    Console.WriteLine(b);
    Console.ReadKey();
}
```

2.5.3 string türüyle ilgili dönüşümler

```
static void Main(string[] args)
{
    Console.WriteLine("İlk Program");
    string b = 3.ToString();
    Console.WriteLine(b);
    Console.ReadKey();
}
```

```
static void Main(string[] args)
{
    Console.WriteLine("İlk Program");
    string b = 12.5f.ToString();
    Console.WriteLine(b);
    Console.ReadKey();
}
```

2.5.4 System.Convert Sınıfı ile Tür Dönüşümü

System isim alanının altındaki Convert sınıfı içinde tür dönüşümü yapabileceğimiz birçok metot bulunur. Ancak bunun için değişken türlerinin CTS karşılıklarını bilmeliyiz. Değişken türleri ve CTS karşılıkları aşağıdaki tabloda listelenmiştir.

Tür	CTS karşılığı
bool	Boolean
byte	Byte
sbyte	Sbyte
short	Int16
ushort	UInt16
int	Int32
uint	UInt32
long	Int64
ulong	UInt64
float	Single
double	Double
decimal	Decimal
char	Char

Convert.ToBoolean(x)
 Convert.ToByte(x)
 Convert.ToSbyte(x)
 Convert.ToInt16(x)
 Convert.ToUInt16(x)
 Convert.ToInt32(x)
 Convert.ToUInt32(x)
 Convert.ToInt64(x)
 Convert.ToUInt64(x)
 Convert.ToSingle(x)
 Convert.ToDouble(x)
 Convert.ToDecimal(x)
 Convert.ToChar(x)

2.5.5 Parse Metodu

Parse, String tipteki verilerin sayısal tiplere dönüşümünü sağlayan methoddur.

```

static void Main(string[] args)
{
    Console.WriteLine("İlk Program");
    string d = "128";
    int c = Int32.Parse(d);
    Console.WriteLine("d=" + d + "\nc=" + c);
    Console.ReadKey();
}

```

Int32.Parse metodunun 3 adet Exception'ı bulunur.

- ArgumentException: Giriş değeri null olduğu zaman gerçekleşir.
- FormatException: Giriş değeri doğru formatta olmadığı zaman gerçekleşir. Örneğin; alfanümerik bir değer gibi.
- OverflowException: Giriş değeri Int32 sınırlarını aştığında gerçekleşir.

Try catch ile exception durumları yakalanabilir.

2.5.6 TryParse

Gönderilen değerin ilgili türe (int32, double, vs) dönüşüp dönüşemeyeğini belirlemektir. TryParse metodu geriye bool tipinde değer döndürür. Eğer parametre olarak gönderilen değer dönüştürülmek istenilen türe başarıyla dönüşürse sonucu out parametresi ile belirtilen parametreye atarır.

```
static void Main(string[] args)
{
    Console.WriteLine("İlk Program");
    string d = "128";
    int a;
    bool c = Int32.TryParse(d, out a);
    Console.WriteLine("d=" + d + "\na=" + a + "\nc=" + c);
    Console.ReadKey();
}
```

2.6 Console Write, WriteLine, Read, ReadLine, ReadKey

Write : Console ekranına çıktı için kullanılır. Write işleminden sonra alt satıra geçilmez. C'deki gibi "\n" ile alt satıra geçilebilir. WriteLine işleminden sonra alt satıra geçilir.

Read: 1 karakter okuma yapar, ReadLine: satır okuma yapar. ReadKey: bir tuşa basılmasını bekler. Basılan tuşun bilgisini verir.

Örnek Uygulama

```
using System;
namespace ConsoleApp1
{
    class Program
    {
        static void Main(string[] args)
        {
            int satir = 50, sutun = 10;
            Console.SetWindowSize(100,20);
            Console.SetCursorPosition(satir, sutun);

            char kr = (char)5;
            Console.Write(kr);
            while (true)
            {
                ConsoleKey K;
                K = (Console.ReadKey().Key);
                if (K == ConsoleKey.UpArrow) {
                    sutun--;
                    Console.Clear();
                }
            }
        }
    }
}
```

```

    }

    if (K == ConsoleKey.DownArrow) {
        sutun++;
        Console.Clear();
    }

    if (K == ConsoleKey.LeftArrow) {
        satir--;
        Console.Clear();
    }

    if (K == ConsoleKey.RightArrow) {
        satir++;
        Console.Clear();
    }

    Console.SetCursorPosition(satir, sutun);
    Console.Write(kr);

    if (K == ConsoleKey.Escape) Environment.Exit(0);
}
}
}
}
}

```

2.7 Rasgele sayı üretimi

Program yazarken sıklıkla rastgele değerlere ihtiyaç duyarız.

Rastgele sayı üretebilmemiz için öncelikle `Random` sınıfı türünden bir nesne üretiriz:

```
Random rasgele=new Random();
```

Buradaki ürettiğimiz nesnenin adı `rasgele`. Şimdi bu nesne üzerinden `Random` sınıfının metotlarına erişiriz:

10 ile 20 arasında int türden rastgele bir sayı üretilir, 10 dâhil ancak 20 dâhil değildir:

```
int RastgeleSayi1=rnd.Next(10,20);
```

0 ile 50 arasında int türden rastgele bir sayı üretilir, 0 dâhil ancak 50 dâhil değildir:

```
int RastgeleSayi2=rnd.Next(50);
```

int türden pozitif herhangi bir sayı üretilir:

```
int RastgeleSayi3=rnd.Next();
```

double türden 0.0 ile 1 arasında rastgele bir sayı üretilir:

```
double RastgeleSayi4=rnd.NextDouble();
```

Random sınıfı System isim alanı içinde bulunduğu için programımızın başında `using System;` satırının bulunması rastgele sayı üretme metotlarını kullanabilmemiz için yeterlidir.

2.8 Diziler

```
int[] dizi=new int[25];
```

veya

```
int[] dizi;  
dizi=new int[25];
```

Yukarıdaki iki kodda da `int` türünden 25 elemanlı dizi adında bir dizi tanımlandı ve dizinin her bir elemanına `int` türünün varsayılan değeri atandı. Varsayılan değerler, sayısal türler için 0, object türü için NULL (yokluk), string türü için "", char için ' ' (boşluk) ve bool için false değerleridir.

Bütün dizilerin birinci elemanı 0. indeksidir. `dizi` dizisinin birinci elemanına `dizi[0]`, 10. elemanına `dizi[9]` yazarak erişebilir ve bu dizi elemanlarını bir değişkenmiş gibi kullanabiliriz.

```
using System;  
class Diziler  
{  
    static void Main()  
    {  
        int[] dizi=new int[20];  
        dizi[5]=30;  
        Console.Write(dizi[5]);  
    }  
}
```

C ve C++ programlama dillerinde olduğu gibi dizilerin elemanlarına aşağıdaki gibi de değer atayabiliriz:

```
string[] dizi1={"Bir","İki","Üç"};  
int[] dizi2={2,-4,6};  
float[] dizi3={2f,1.2f,7f};
```

Ancak bu şekilde dizi belirtimini sadece dizi tanımlamalarında kullanabiliriz. Aşağıdaki kullanım da hatalıdır:

```
int[] dizi;  
dizi={1,2,3};
```

Dizi belirtimi sadece dizi tanımlamalarında geçerlidir. Dizi tanımlaması şu şekilde de yapılabilir:

```
int[] dizi=new int[]{1, 2, 3};
```


2.9 ArrayList

ArrayList sınırları dinamik olarak değişebilen diziler olarak tanımlanır. Bu veri yapısı .NET sınıf kütüphanesinin **System.Collections** isim alanında bulunur.

Koleksiyon sınıfları özel tasarlanmış nesneleri ve onlara ait olan görevleri yerine getirmek için oluşturulmuş olan nesnelerdir.

2.9.1 ArrayList Metotları

- **Add (Ekle):** Bu metot ArrayList'in sonuna yeni bir nesne ekler
- **Count:** Count ArrayList'in boyutunu verir. Ancak, elemanları teker teker sayarak yapmaz, bunun yerine her işlemten sonra değeri güncellenir. Yani bir eklemede değeri bir artar, bir çıkarmada değeri bir azalır. Bu açıdan hızlı sonuç verir.
- **Clear:** ArrayList'in tüm elementlerini siler.
- **Sort:** Sıralama işlemini yapar.
- **Reverse:** Terse çevirme işlemi yapar.
- **GetRange:** ArrayList'in istenilen aralıkta veriyi çekebilir ve yeni bir ArrayList döndürür. Sadece ArrayList'in bir kısmı ile ilgilendiğimizde ideal bir metottur.
- **SetRange:** Mevcut ArrayList'in bir kısmını seçmede kullanılır. GetRange sonuçları yeni bir ArrayList taşırken, SetRange taşımaz.
- **IndexOf:** Aranan bir element için IndexOf ilk bulduğu değerin pozisyonunu döndürür. Eğer elementi bulamazsa -1 döndürür.
- **LastIndexOf:** Aranan bir element için son sonucu bulur.
- **Convert:** ArrayList'inizi Array veri türüne dönüşüm yapabilirsiniz.

```

1  using System;
2  using System.Collections;
3
4  namespace ConsoleApp1
5  {
6      class Program
7      {
8          static void Main(string[] args)
9          {
10             ArrayList listem = new ArrayList();
11             listem.Add("Bir");
12             listem.Add("İki");
13             listem.Add("Üç");
14
15             for (int i=0; i<listem.Count; i++)
16             {
17                 Console.WriteLine(i+ ": " + listem[i]);
18             }
19
20             int j = 0;
21             foreach(string i in listem)
22             {
23                 Console.WriteLine(j + ": " + listem[j]);
24                 j++;
25             }
26         }
27     }
28 }

```

2.10 List

List, `System.Collections.Generic` isim uzayı içinde tanımlanmış metodlar, özellikler ve diğer sınıflarda olduğu gibi `insert`, `remove`, `search` vb. nesneleri barındırmaktadır.

`ArrayList`'e benzer bir yapıdır ancak veri türü "<" ">" içinde tanımlanır.

Liste tanımlama:

```

List<int> asalSayilar = new List<int>();
    asalSayilar.Add(1);
    asalSayilar.Add(3);
    asalSayilar.Add(5);
    asalSayilar.Add(7);

```

Eleman Ekleme: `Insert()` metodunu kullanarak listeye eleman ekleyebilirsiniz. Bu metotla liste içinde herhangi bir indexe ekleme yani aralara da ekleme yapabilirsiniz.

Eleman Silme:

```
var numbers = new List<int>(){ 10, 20, 30, 40, 10 };  
numbers.Remove(10); // 10 sayısını siler.  
numbers.RemoveAt(2); //3. elemanı siler (index numarası 0 dan başlar.)
```

Liste İçinde Arama:

```
if (isimler.Contains("Ahmet")) {  
    MessageBox.Show("Ahmet bu listede mevcuttur.");  
}
```

2.11 Region

Kodları daha düzenli olarak yazmak ve kodların okunurluğunu arttırmak için kullanılır.

Region eklenmek istenilen kodun başında #region, sonunda #endregion ifadesi yazılır.

2.12 Console Uygulamasına Argüman Gönderme

Uygulamamızı komut satırından çağırırken exe dosyamızın yanında parametreler gönderebiliriz.

```
static void Main(string[] args)  
{  
    foreach (string aaa in args)  
    {  
        Console.WriteLine("Dışardan gönderilen argümanlar: {0}" , aaa);  
    }  
}
```