İsim Alanları (NameSpaces) ve System İsim Alanı

- İsim alanları, yazılan programlarda mantıksal organizasyonu sağlar.

- Eski programlama dillerinde, çok kişi tarafından yazılan projelerde, isim çakışmaları meydana gelebilmekteydi. Bu çakışmalar fonksiyon, alt program ya da sınıf isimlerinin değiştirilmesi ile çözülebiliyordu.

- Bu çözüm sonucunda farklı kişiler ya da firmalar tarafından yazılan aynı isimli metotların firma1Menu(), firma2Menu() vb. şeklinde isimlendirilmesine neden olur.

- Bu çözüm çok kullanıldıkça fonksiyon isimlerinin önüne anlamsız bir çok ekler gelecektir.

- Bu karmaşanın önüne geçebilmek için modern dillerde mantıksal bir ayırma yapısı kullanılmaya başlanmıştır.

- C++ ve C# dillerindeki isim alanları (namespace) ya da Java dilindeki paketler (package) buna birer örnektir.

- Bu yapılar sayesinde aynı isimde sınıflar, yapılar kolaylıkla tanımlanabilir ve daha önceki tanımlananlarla herhangi bir soruna neden olmaz.

- Programlara isim alanları using anahtar sözcüğü ile bildirilir.

- İsim alanlarının bildirilmesi zorunlu olmamakla birlikte sıkıcı tekrarları ortadan kaldırır.

- İsim alanı bildirilmediğinde sınıfın, değişkenin ya da metodun tam ismiyle kullanılması gerekmektedir.

```
class Program
{
    static void Main()
    {
        System.Console.WriteLine("Tam isim");
    }
}
```

Ya da →

```
using System;

class Program
{
    static void Main()
    {
        Console.WriteLine(“using ile");
    }
}
```

- Temel veri türlerini kullanmak için System isim alanını eklemek gerekmez. C# derleyicisi bunları otomatik olarak System isim alanını kullanarak işletir.

```
class Program
{
    static void Main()
    {
        int a = 10; // Doğru
        Int32 b = 20; // Yanlış
    }
}
```

İsim Alanı Bildirimi

- İsim alanı namespace anahtar sözcüğü ile bildirilir.

- Namespace için yine sınıflarda olduğu gibi bir parantez blok açılır. Bu bloğa ise isim alanı içinde yer alacak bildirimler tanımlanır.

```
using System;

namespace PrgDil3
{
    class Deneme
    {
        public int a;
        public int b;

        public Deneme(int a, int b)
        {
            this.a = a;
            this.b = b;
        }

        public static int Topla(int x, int y)
        {
            return x + y;
        }
    }
}

class Program
{
    static void Main()
    {
        PrgDil3.Deneme d = new PrgDil3.Deneme(1,2);
        Console.WriteLine(d.a);
        Console.WriteLine(PrgDil3.Deneme.Topla(2, 3));
    }
}
```

- Farklı konumlarda ya da dosyalarda aynı adlı isim alanları tanımlanabilir. Bu hataya neden olmaz. Çünkü isim alanları mantıksal yapılardır.

- İsim alanları genellikle kod organizasyonu ve isim çakışmalarını engellemek için kullanılır.

- İsim alanları içersinde yalnızca sınıf (class), numaralandırma (enum), yapı (struct), temsilci (delegate) ya da arayüz (interface) bildirimi yapılabilir.

- İsim alanları içinde değişken tanımlaması ve ya metot bildirimi yapılamaz.

```
using System;

namespace isimalan1
{
    class Deneme
    {
        public Deneme()
        {
            Console.WriteLine("Deneme 1");
        }
    }
}

namespace isimalan2
{
    class Deneme
    {
        public Deneme()
        {
            Console.WriteLine("Deneme 2");
        }
    }
}

class Program
{
    static void Main()
    {
        isimalan1.Deneme d1 = new isimalan1.Deneme();
        isimalan2.Deneme d2 = new isimalan2.Deneme();
    }
}
```

- Tanımlanan isim alanları using ifadesi ile belirtilebilir.

- using anahtar sözcüğü tanımlamaların üstünde yer almalıdır. Bu şekilde tanımlandıktan sonra isim alanı içersindeki türlere tam isim belirtmeden erişilebilir.

```
using System;


namespace isimalan1
{
    class Deneme
    {
        public Deneme()
        {
            Console.WriteLine("Deneme 1");
        }
    }
}


class Program
{
    static void Main()
    {
        Deneme d1 = new Deneme();
     }
}
```

- using anahtar sözcüğü ile takma isim (alias) vermek de mümkündür.

- Bazı durumlarda using ile tanımlanmış iki isim alanı içersinde aynı isimli sınıflar olabilir. Bu durumda doğrudan sınıf isminin kullanılması hataya neden olur. Çünkü derleyici, iki isim alanı içersinde yer alan aynı isimli sınıflardan hangisinin kastedildiğini anlayamaz.

- Bunu aşmak için takma isimler tanımlanabilir.

```
using System;
using isimalan1;
using isimalan2;

using Den1 = isimalan1.Deneme;
using Den2 = isimalan2.Deneme;
```

```
using System;
using isimalan1;
using isimalan2;

using Den1 = isimalan1.Deneme;
using Den2 = isimalan2.Deneme;

namespace isimalan1
{
    class Deneme
    {
        public Deneme()
        {
            Console.WriteLine("Deneme 1");
        }
    }
}

namespace isimalan2
{
    class Deneme
    {
        public Deneme()
        {
            Console.WriteLine("Deneme 2");
        }
    }
}

class Program
{
    static void Main()
    {
        Den1 d1 = new Den1();
        Den2 d2 = new Den2();
    }
}
```

İç İçe Geçmiş İsim Alanları Tanımlamak (Nested NameSpaces)

- İsim alanları içersinde başka isim alanları da tanımlanabilir. Bu şekilde hiyerarşik düzenlemeler yapılır.

- Bu şekilde tanımlanmış isim alanlarına "." operatörü ile erişilir. using ile tanımlanırken de aynı yöntem kullanılır.

```
using System;

namespace Alan
{
    class Sinif1
    {
        public Sinif1() { }
    }

    namespace AltAlan
    {
        class Sinif2
        {
            public Sinif2() { }
        }
    }
}

class Program
{
    static void Main()
    {
        Alan.Sinif1 s1 = new Alan.Sinif1();
        Alan.AltAlan.Sinif2 s2 = new Alan.AltAlan.Sinif2();
    }
}
```

Harici Takma İsimler (External Alias)

- Daha önce bahsedildiği gibi, farklı isim alanları altında aynı isimli sınıflar tanımlandığında ve her iki isim alanı da using deyimi ile programa eklendiğinde kod bloğu içersinden sınıflara erişmek istediğimizde takma isimler tanımlamak gerekiyordu.

  Böylelikle referans edilmiş her bir sınıfa takma ismi yardımıyla erişebilmekteydik.

- Bu tarz bir çözüm büyük program parçalarında anlam bütünlüğünün bozulmasına neden olabilir.

- Bunun için C# 2.0'dan itibaren C# dilinde oluşturulmuş olan sınıf kütüphanelerine (dll'ler) harici takma isimler verebilmekteyiz.

Örnek

- Farklı iki firmadan benzer işler yapan iki kütüphane satın alınsın. Örneğin A firmasından 3 boyutlu çizim işlemleri yapan bir kütüphane, B firmasından ise 2 boyutlu çizim işlemleri yapan bir kütüphane alınsın.

- Her iki kütüphanede de Grafik isimli bir isim alanı ve bu isim alanının altında da Nokta, Kare, Ucgen, Dikdortgen gibi sınıflar bulunmaktadır. İsim alanları ve sınıflar aynı olduğundan herhangi bir sınıfı kullanmaya kalktığımızda çakışma olacaktır.

Bu gibi durumlarda çakışmaların önüne geçebilmek için daha önce de belirtildiği gibi sınıf kütüphaneleri tanımlanmalıdır.

A firmasının kütüphanesi a.cs olsun.

```
using system;
Namespace Grafik
{ public class Nokta
     {....
      ....}
 public class Ucgen
    {....
      ....}
     .... ....
     .... ....
}
```

B firmasının kütüphanesi b.cs olsun.

```
using system;
Namespace Grafik
{ public class Nokta
     {....
      ....}
 public class Ucgen
    {....
      ....}
     .... ....
     .... ....
}
```

- Her iki kütüphaneyi dll haline getirmek için komut satırından aşağıdaki komutu ayrı ayrı çalıştırmamız gerekir.

```
csc /t:library  a.cs
csc /t:library b.cs
```

- Yukarıdaki her iki sınıf kütüphanesi derlendikten sonra aynı dizin içerisinde sınıf kütüphanelerimize ait olan dll dosyaları oluşmuş olur.

Sınıf kütüphanelerini bu şekilde derledikten sonra ana program bloğumuzda kullanabilir miyiz?

- Bu yapıyı bu şekilde ana program bloğumuzda kullanamayız çünkü kullanmaya kalktığımızda herhangi bir alias belirtmediğimiz için hataya sebep olur.

Derleme işleminde takma isimler belirtmek için ana programın derleme işlemini komut satırından yapıyor isek aşağıdaki komutu işletmemiz gerekir.

```
csc /r:firma1=a.dll /r:firma2=b.dll program cs
```

Programımızı bu şekilde derledikten sonra ana program bloğumuz içerisinde artık aşağıdaki gibi ifadelerin kullanılması geçerli olacaktır.

```
firma1::Grafik.Nokta();
firma1::Grafik.Ucgen();

firma2::Grafik.Nokta();
firma2::Grafik.Ucgen();
```

System İsim Alanı

- .NET sınıf kütüphanesinde yer alan System isim alanı içersinde oldukça kullanışlı bazı sınıflar bulunmaktadır.

- Bunlardan System.Array, System.Random, System.Convert ve System.GC ’den daha önce bahsedilmişti.

- Diğer önemli sınıflar şunlardır:

System.Math

- .NET sınıf kitaplığında bazı temel matematiksel işlemleri yapmak için static metotlar ve bazı özellikler içeren sınıftır.

- Static tanımlı metotlar içerdiğinden nesne tanımlamaya gerek kalmadan Math.MetotAdı(parametreler) şeklinde erişilip kullanılabilir.

- Math sınıfının 2 tane özelliği vardır. Bunlar matematikte sıkça kullanılan pi ve e sayılarıdır.
  - Math.PI
  - Math.E

  şeklinde tanımlanmışlardır.

```
using System;

class Program
{
    static void Main()
    {
        Console.WriteLine(Math.PI);
        Console.WriteLine(Math.E);
    }
}
```

Math sınıfının bazı önemli metotları ise şu şekildedir.

Abs(x)	**x sayısının mutlak değerini alır.**
Cos(x)	**x değerinin kosinüsünü hesaplar.**
Sin(x)	**x değerinin sinüsünü hesaplar.**
Tan(x)	**x değerinin tanjantını hesaplar.**
Ceiling(x)	**x sayısını x'ten büyük en küçük tam sayıya yuvarlar.**
Round(x)	**x sayısını noktadan sonraki küsüratı 0.5 den küçük ise bir alt; 0.5 e eşit ve büyük ise bir üst tam sayıya yuvarlar.**
Floor(x)	**x sayısını x'ten küçük en büyük tam sayıya yuvarlar.**
Max(x,y)	**x ve y sayılarından büyüğünü bulur.**
Min(x,y)	**x ve y sayılarından küçüğünü bulur.**
Pow(x,y)	**x'in y'ninci kuvvetini hesaplar.**
Sqrt(x)	**x'in karekökünü hesaplar.**
Log(x)	**x değerinin e tabanına göre logaritmasını hesaplar.**
Log10(x)	**x değerinin 10 tabanına göre logaritmasını hesaplar.**
Exp(x)	**e üzeri x değerini hesaplar.**

```
using System;

class Program
{
    static void Main()
    {
        Console.WriteLine(Math.Abs(-5.75));
        Console.WriteLine(Math.Cos(60));
        Console.WriteLine(Math.Sin(60));
        Console.WriteLine(Math.Tan(45));
        Console.WriteLine(Math.Ceiling(5.4));
        Console.WriteLine(Math.Floor(5.6));
        Console.WriteLine(Math.Round(5.5));
        Console.WriteLine(Math.Max(-5.75,-5.25));
        Console.WriteLine(Math.Min(8,3));
        Console.WriteLine(Math.Pow(2,20));
        Console.WriteLine(Math.Sqrt(65536));
        Console.WriteLine(Math.Log(5 * Math.E));
        Console.WriteLine(Math.Log10(100));
        Console.WriteLine(Math.Exp(1));
    }
}
```

System isim alanı içersinde yer alan temel veri tiplerinin sahip olduğu bazı metotlar da zaman zaman kullanılmaktadır.

- Tip.Parse(); : string biçimindeki verileri tip türüne çevirir.
- Nesne.CompareTo(object o); : metodu çağrılan nesne ile o nesnesini karşılaştırır. Değerler eşit ise 0, nesne değeri küçük ise negatif, büyük ise pozitif değer döndürür.
- Nesne.EqualTo(object o); : metodu çağrılan nesne ile o nesnesini karşılaştırır. Eşit ise true aksi halde false döndürür.
- Nesne.ToString(); : nesne değerini string şeklinde geri döndürür.

- Double ve float değerler için bir önceki anlatılan metotlara ek olarak aşağıdaki metotlar da bulunur. Bu metotlar true/false yani boolean geri dönüş değerine sahiptir:

  - IsInfinity: Sayının sonsuzu temsil edip etmediğini kontrol eder.
  - IsNaN: Anlamlı bir sayı olup olmadığını kontrol eder.
  - IsPositiveInfinitiy, IsNegativeInfinity: Sayının + ya da – sonsuz olup olmadığını kontrol eder.

- Char ve string veriler içinde bazı metotlar bulunur:

  - GetNumericValue : Eğer değer sayısal karakter içeriyorsa sayısal değeri aksi halde -1 döndürür.

  - IsControl, IsDigit, IsLetter, IsLetterOrDigit, IsLower, IsNumber, IsPunctuation, IsSeperator, IsSurroGate, IsSymbol, IsUpper, IsWhiteSpace gibi statik metotlar da değerin çeşitli değer kümesi içinden olup olmadığını test eder. (rakam mı, küçük harf mi sembol mü vs.)

- Decimal değerler içinde bazı metotlar mevcuttur:
  - Add (+), Divide(/), Multiply(*), Subtract(-), Remainder(%) gibi aritmetiksel işlem yapan metotlar.
  - Floor, Round gibi yuvarlama metotları.
  - Negate sayının negatifini bulma
  - GetBits sayının bitlerini elde etme
  - Truncate sadece tam sayı kısmını alma

gibi metotlara sahiptir.

DataTime ve DateSpan

- C# dilinde tarih ve saat işlemleri System isim alanında bulunan DateTime ve TimeSpan yapıları ile gerçekleştirilir.

- DateTime yıl, ay, gün, saat, dakika, saniye gibi bilgileri tutan bir yapıdır.

- TimeSpan ise iki zaman bilgisi arasındaki farkı temsil etmek için kullanılır.

```
using System;

class Program
{
    static void Main()
    {
        Console.WriteLine(DateTime.MinValue);
        Console.WriteLine(DateTime.MaxValue);
        Console.WriteLine(DateTime.Now);
        Console.WriteLine(DateTime.Today);

        DateTime Bugun = new DateTime();

        Bugun = DateTime.Now;

        Console.WriteLine(Bugun.Date);
        Console.WriteLine(Bugun.Day);
        Console.WriteLine(Bugun.Month);
        Console.WriteLine(Bugun.Year);
        Console.WriteLine(Bugun.DayOfYear);
        Console.WriteLine(Bugun.DayOfWeek);
        Console.WriteLine(Bugun.TimeOfDay);
        Console.WriteLine(Bugun.Hour);
        Console.WriteLine(Bugun.Minute);
        Console.WriteLine(Bugun.Second);
        Console.WriteLine(Bugun.Millisecond);
        Console.WriteLine(Bugun.Ticks);
    }
}
```

- TimeSpan ve DateTime yapıları ile tanımlanmış bazı operatörler bulunur.

- İki tarih arasındaki farkı bulmak için çıkarma(-), ileriki bir tarihi hesaplamak için toplama(+), iki tarih arasında büyüklük küçüklük karşılaştırması yapmak için de “<” ve “>” operatörleri aşırı yüklenmiştir.

```
using System;

class Program
{
    static void Main()
    {
        int yil, ay, gun;
        Console.Write("Doğum Yılınız:");
        yil = Convert.ToInt32(Console.ReadLine());
        Console.Write("Doğum Ayınız:");
        ay = Convert.ToInt32(Console.ReadLine());
        Console.Write("Doğum Gününüz:");
        gun = Convert.ToInt32(Console.ReadLine());

        DateTime Bugun = DateTime.Today;
        DateTime DogumGunu = new DateTime(yil, ay, gun);

        TimeSpan fark = Bugun - DogumGunu;

        Console.WriteLine("Doğduğunuz Gün: {0}", DogumGunu.DayOfWeek);
        Console.WriteLine("Gün sayısı: {0}", fark.Days);

        Console.WriteLine();
        Console.Write("Gün sayısı:");
        gun = Convert.ToInt32(Console.ReadLine());

        TimeSpan GunSayisi = new TimeSpan(gun, 0, 0, 0);
        DateTime Gelecek = DateTime.Today + GunSayisi;
        Console.WriteLine("{0} gün sonra günlerden {1} dir.", gun, Gelecek.DayOfWeek);
    }
}
```

System.BitConverter

- Alt seviye programlama da veriler byte dizisi şeklinde yani bitsel olarak işlenir.
- .NET içersinde de bite çevirmek amacıyla BitConverter sınıfı bulunur.
- Bu sınıfın IsLittleEndian isimli bi özelliği bulunur. Bu özellik işlemci mimarisine göre verileri bellekte depolarken hangi sıralamada yerleştirildiğini kontrol eder.
- En önemli metodu da GetBytes'tır. Amacı da farklı sayı türlerini byte dizisine çevirir.

```
using System;

class Program
{
    static void Main()
    {
        if (BitConverter.IsLittleEndian)
            Console.WriteLine("Little Endian");
        else
            Console.WriteLine("Big Endian");

        int a = 46513;

        byte[] b = BitConverter.GetBytes(a);

        foreach (byte x in b)
            Console.WriteLine(x);
    }
}
```

System.Buffer

- Buffer sınıfı ile tür bilgisinden bağımsız bir biçimde byte düzeyinde veri işleme yapılır. Dizilerin belirli alanları başka bir diziye tür bilgisine bakılmaksızın aktarılabilir. Tüm veriler byte dizisi şeklinde düşünülür.

- BlockCopy, GetByte, SetByte en önemli metotlarıdır.

```
using System;

class Program
{
    static void Main()
    {
        byte[] kaynak = { 1, 2, 0, 1 };
        short[] hedef = new short[5];

        Buffer.BlockCopy(kaynak, 0, hedef, 0, 4);

        foreach (short s in hedef)
            Console.Write(s + " ");

        Console.WriteLine("\n" + Buffer.GetByte(hedef, 0));
        Buffer.SetByte(hedef, 5, 3);

        foreach (short s in hedef)
            Console.Write(s + " ");

        Console.WriteLine();
        Console.WriteLine(Buffer.ByteLength(kaynak));
        Console.WriteLine(Buffer.ByteLength(hedef));
    }
}
```