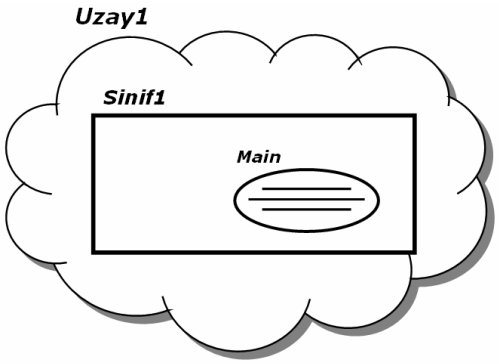
# SINIFLAR

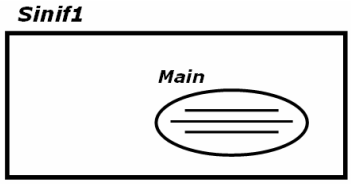
## İsim Uzayı Kavramı

C# programlama dilinde sınıflar, isim uzayları içerisinde yer alırlar. Her ne kadar bir C# programının derlenerek çalıştırılabilmesi için programın kaynak kodunda yer alan sınıfların bir isim uzayı içerisine yerleştirilme zorunluluğu olmasa da, programın tasarımının doğallığı, anlaşılabilirliği ve programa hiyerarşik bir yapı kazandırma kolaylığı bakımından benzer amaçlar için hazırlanmış ve/veya ortak yanları olan sınıfların uygun bir isme sahip bir isim uzayı içerisine konmaları doğru bir yaklaşımdır.

```
using System;
// isim uzayi kullanilmis
namespace Uzay1
{
    class Sinif1
    {
        static void Main()
        {
            Console.WriteLine("Merhaba dunya!");
            Console.ReadLine();
        }
    }
}
```

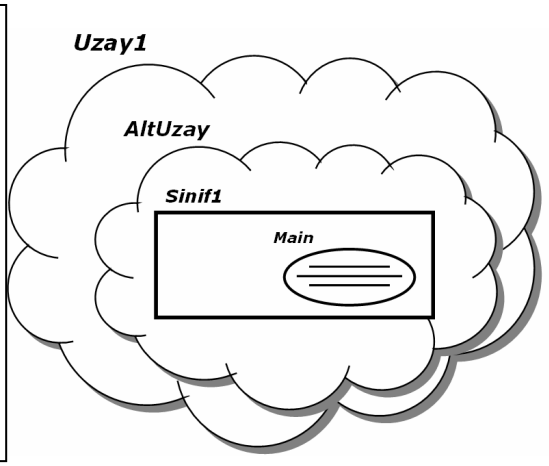



```
using System;
// isim uzayi kullanilmamis
class Sinif1
{
    static void Main()
    {
        Console.WriteLine("Merhaba dunya!");
        Console.ReadLine();
    }
}
```

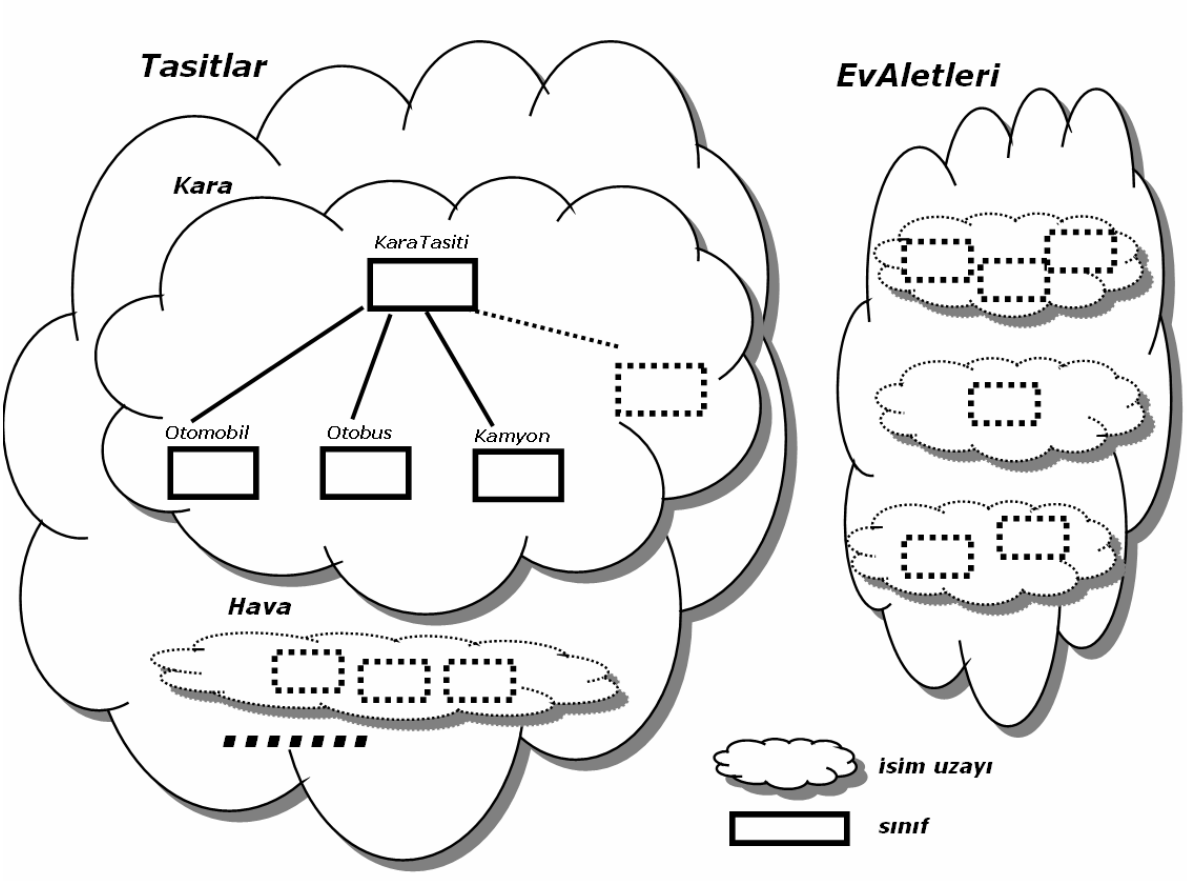



Yukarıda yer alan iki farklı C# programından birincisinde isim uzayı kullanılmış, ikincisinde ise kullanılmamıştır. Bu programlardan her ikisi de sorunsuz bir biçimde derlenip çalıştırılabilmektedirler. Hatta görevi sadece ekrana "**Merhaba dünya!**" yazdırmak olan, tek bir sınıf içerisinde bir **Main** fonksiyonundan ibaret olan basit bir program için bir isim uzayı belirtilmesi lüzumsuz dahi görülebilir. Ancak çok daha geniş çaplı uygulamaların tasarlanması söz konusu olduğunda hiyerarşik yapıya duyulan ihtiyaç artacak ve sınıfların isim uzayları içerisinde konumlandırılması bir ihtiyaç haline gelecektir.

```
using System;
// ic ice isim uzaylari
namespace Uzay1
{
    namespace AltUzay
    {
        class Sinif1
        {
            static void Main()
            {
                Console.WriteLine("Merhaba dunya!");
                Console.ReadLine();
            }
        }
    }
}
```




Yukarıdaki örnek programda görüldüğü gibi, isim uzayları iç içe de kullanılabilirler.




Yukarıdaki şekilde, isim uzayları ile sınıfların bir arada (iç içe) kullanımı, görsel olarak anlatılmaya çalışılmıştır. Verilen örnekte, insanların günlük hayatta yararlandıkları materyallerin hiyerarşik bir yapı aracılığı ile temsil edilmesi ele alınmaktadır.

Öncelikle, "**sınıf**" kavramı ile "**isim uzayı**" kavramını birbiri ile karıştırmamak gerekir. İsim uzayları, programlama yaparken neyin nerede olduğunu kolayca hatırlatmaktan ve birbiri ile

ilişkisi olan sınıfları bir arada tutmaktan çok da öteye gidemezler. Oysa sınıf yapısı, nesne yönelimli tasarımın (ve dolayısı ile nesne yönelimli programlamanın) can alıcı noktasını teşkil etmektedir.

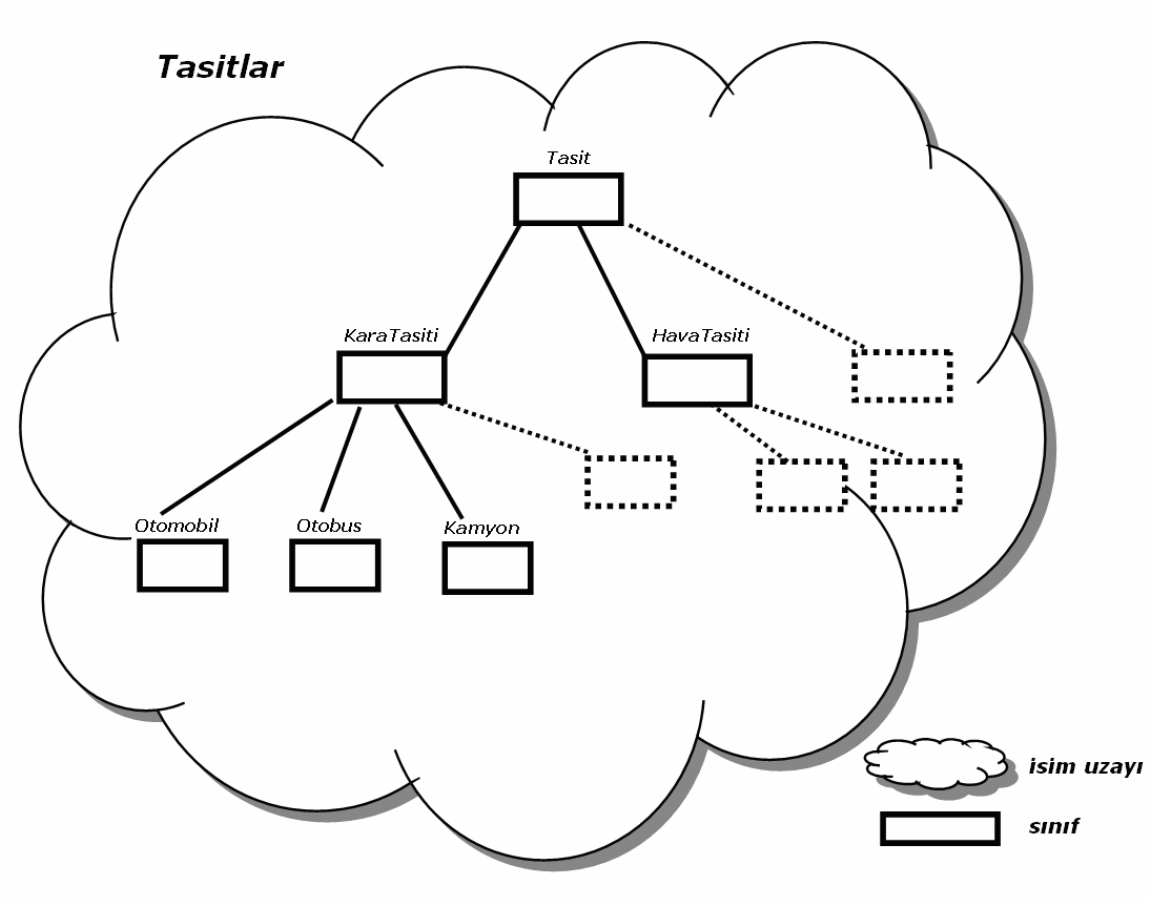
Sözgelimi, A ülkesinin kendi vatandaşlarına sunmakta olduğu vatandaşlık haklarından, B ülkesinin bir vatandaşı yararlanamaz. Her insanın, dünyadaki bütün ülkelerin vatandaşlık haklarından sınırsızca yararlanabilmesi, istenen bir durum değildir. Benzer biçimde, nesne yönelimli programlamada da bir sınıfın, başka bir sınıfa ait tüm fonksiyonlardan yararlanabilmesi, bu sınıfın tüm üyelerine erişebilmesi istenmeyen bir durumdur. Nesne yönelimli programlamanın ortaya çıkış amacı da, günlük hayattaki yetkilendirme, miras alma, hiyerarşik yapı, belirli bir şablon oluşturarak defalarca kullanabilme gibi pek çok eylemi programlama ortamına dökerek insanlara daha doğal gelen, mantığa daha uygun bir program geliştirme tarzı sunmaktır.

Yukarıda çizimsel olarak verilen örneğe geri dönecek olursak, günlük hayatta etkileşim içerisinde olduğumuz materyaller arasında neredeyse hiçbir bağı olmayan, birbiriyle tamamen ayrık iki küme gibi düşünülebilecek "**Taşıtlar**" ve "**Ev Aletleri**" (Çizimde Türkçe karakter ve boşluk kullanılmadan isimlendirme yoluna gidilmiştir.), iki farklı isim uzayını oluşturmaktadırlar.

Kara taşıtlarının da hava taşıtlarıyla ortak yönü olmadığı düşünülerek, bunlar da "**Taşıtlar**" isim uzayı içerisine iki ayrık alt isim uzayı olarak verilmiştir. Ancak kara taşıtlarının kendi aralarında yakın ilişki içerisinde bulunabilecekleri düşünülerek burada sınıf yapısından yararlanılmak istenmiştir. Zaten bir program elde edebilmek için sınıf yapısı kurarak içerisini doldurmak şarttır. Sınıf yapıları, isim uzayları gibi programlama kolaylığı sağlayan sembolik yapılardan çok daha öte oldukları için, hangi sınıfın hangi üyelerinin erişim haklarının ne olacağı, bu sınıflar arasındaki kalıtımsal ilişkilerin ne şekilde olacağı, gerçeklenmekte olan programın hangi amaca hizmet edeceği de göz önünde bulundurularak dikkatle belirlenmelidir. Verilen örnekte her ne kadar **Kara Taşıtı** sınıfı ve **Otomobil** sınıfı aynı ad uzayında yer alsalar da **Otomobil** sınıfı, **Kara Taşıtı** sınıfından kalıtlamaktadır (Kalıtlama kavramı ilerleyen konularda daha ayrıntılı olarak ele alınacaktır.). Yani, **Otomobil** sınıfının her bir nesnesi, **Kara Taşıtı** sınıfında yer alan ve **Kara Taşıtları** sınıfından kalıtlayan sınıfların nesnelerinin erişmesine müsaade edilen tüm **Kara Taşıtları** sınıfı elemanlarına

erişebileceklerdir; ancak tersi bir kalıtlama ilişkisi söz konusu olmadığından, **Kara Taşıtları** sınıfına ait nesneler için **Otomobil** sınıfının elemanlarına erişmekten bahsedilemez. Bu anlatımı, *"Her otomobil bir kara taşıtıdır ancak her kara taşıtı bir otomobil değildir."* ifadesi ile de destekleyebiliriz.

Tabi ki bu hiyerarşik yapı, programcının yaklaşımına göre şekillendirilecektir. Örneğin programcı, kara ve hava taşıtları arasındaki ilişkilerden, benzerliklerden yararlanmak istiyorsa, bunlar için iki ayrık isim uzayı tasarlamak yerine "**Tasit**" adında bir genel sınıf ve bu sınıflan kalıtlayan alt sınıflar oluşturabilir (aşağıdaki örnekte olduğu gibi).




Gerekli görüldüğünde, başka bir isim uzayında yer alan sınıflara ve bu sınıfların elemanlarına erişmek de mümkündür:

```
namespace Uzay1
{
    class Sinif1
    {
        static void Main()
        {
            System.Console.WriteLine("Merhaba dunya!");
            System.Console.ReadLine();
        }
    }
}
```

Yukarıdaki programın 7. ve 8. satırlarında, **System** isim uzayı içerisinde bulunan **Console** sınıfına ait statik erişimli birer fonksiyon olan "**WriteLine**" ve "**ReadLine**" fonksiyonları çağrılmıştır. Görüldüğü gibi, bu fonksiyonlara ulaşılabilmek için öncelikle bu fonksiyonların bulunduğu sınıfın da yer aldığı isim uzayı, sonra ise bu fonksiyonların yer aldığı sınıf belirtilmektedir.

```
using System;

namespace Uzay1
{
    class Sinif1
    {
        static void Main()
        {
            Console.WriteLine("Merhaba dunya!");
            Console.ReadLine();
        }
    }
}
```

Yukarıdaki programda ise 1. satırda yer alan "**using System**" deyimi, "**System** isim uzayında yer alan bütün sınıfları görünür kıl." anlamını taşıdığı için, 8. ve 9. satırlarda "**WriteLine**" ve "**ReadLine**" fonksiyonlarının çağrılmasında isim uzayının belirtilmesine lüzum kalmamış, bu fonksiyonların yer aldığı sınıfların belirtilmesi yeterli olmuştur.

```
using System;

namespace Tasitlar
{
    class Deniz
    {
        public static void Yazdir()
        {
            Console.WriteLine("Tasitlar > Deniz > Yazdir");
        }
    }
}

namespace Seyahat
{
    class Deniz
    {
        public static void Yazdir()
        {
            Console.WriteLine("Seyahat > Deniz > Yazdir");
        }
    }
}

namespace Genel
{
    class Sinif1
    {
        static void Main()
        {
            Tasitlar.Deniz.Yazdir();
            Seyahat.Deniz.Yazdir();
            Console.WriteLine("Genel > Sinif1 > Main");
            Console.ReadLine();
        }
    }
}
```

*Çıktı:*

```
Tasitlar > Deniz > Yazdir
Seyahat > Deniz > Yazdir
Genel > Sinif1 > Main
```

C# ile yazdığımız konsol uygulamaları, bir ya da birden çok sayıda kaynak kod dosyasından oluşan projelerdir. Tek bir kod dosyasında birden çok sayıda isim uzayı bulunabilir. Bu isim uzaylarının içerisinde aynı isimde sınıfların bulunması mümkündür. Bu açıdan bakıldığında isim uzaylarının, aynı isme sahip çok sayıda sınıf tanımlanmasını mümkün kıldığı, yani sınıflar arasında isim çakışmasını önlediği söylenebilir.

Ayrıca belli bir isim uzayını birden çok sayıda kod dosyasında kullanmak da mümkündür. Aşağıda verilen örnekte, "**Kaynak1.cs**" ve "**Kaynak2.cs**" isimlerinde iki farklı kod dosyası vardır ve bunların her ikisinde de **Seyahat** isim uzayı yer almaktadır. Programı bu şekilde hazırlamakla, birinci kod dosyasında bulunan **Hava** sınıfını, ikinci dosyada yer alan **Seyahat** isim uzayının içerisine yerleştirmek arasında mantıksal olarak fark yoktur.

***Kaynak1.cs***

```
using System;

namespace Seyahat
{
    class Hava
    {
        public static void Yazdir()
        {
            Console.WriteLine("Seyahat > Hava > Yazdir");
        }
    }
}
```

***Kaynak2.cs***

```
using System;

namespace Tasitlar
{
    class Deniz
    {
        public static void Yazdir()
        {
            Console.WriteLine("Tasitlar > Deniz > Yazdir");
        }
    }
}

namespace Seyahat
{
    class Deniz
    {
        public static void Yazdir()
        {
            Console.WriteLine("Seyahat > Deniz > Yazdir");
        }
    }
}

namespace Genel
{
    class Sinif1
    {
        static void Main()
        {
            Tasitlar.Deniz.Yazdir();
            Seyahat.Deniz.Yazdir();
            Seyahat.Hava.Yazdir();
            Console.WriteLine("Genel > Sinif1 > Main");
            Console.ReadLine();
        }
    }
}
```

*Çıktı:*

```
Tasitlar > Deniz > Yazdir
Seyahat > Deniz > Yazdir
Seyahat > Hava > Yazdir
Genel > Sinif1 > Main
```

## Sınıf Yapısı

Sınıf yapısı, nesne yönelimli tasarım ve programlama kavramı ile doğrudan ilişkilidir. Nesneler sınıflardan türetilirler ve türetildikleri sınıfın özelliklerini taşırlar. C# programlarının giriş kapısı olan **Main** fonksiyonu da dâhil olmak üzere tüm fonksiyonlar, bir sınıf içerisinde tanımlanmak zorundadırlar. Bu bakımdan sınıf yapısı programlar için bir zorunluluk olmaktadır ve bu zorunluluk da C# programlama dilinin nesne yönelimli bir yapıya sahip olmasıyla ifade edilebilir.

Daha önce sınıfların içerisinde yer alan üyelerle (değişken, fonksiyon) kullanımlarını incelemiş olduğumuz erişim belirleyicileri (**public**, **private**, **protected**; **static**), sınıfların kendilerine olan erişim hak ve biçimlerinin belirlenmesinde de rol almaktadırlar. Ayrıca sınıflar için, ilgili C# uygulaması kapsamındaki diğer sınıflar tarafından erişilebilirliği mümkün kılan "**internal**" erişim belirleyicisi de söz konusudur.

```
using System;

namespace Uzay1
{
    class Sinif1
    {
        public static int deger = 6;

        public static void Fonksiyon1()
        {
            Console.WriteLine("Sinif1 > Fonksiyon1");
        }
    }

    class Sinif2
    {
        static void Main()
        {
            Console.WriteLine("Sinif1 > deger : {0}", Sinif1.deger);
            Sinif1.Fonksiyon1();
            Console.ReadLine();
        }
    }
}
```

*Çıktı:*

```
Sinif1 > deger : 6
Sinif1 > Fonksiyon1
```

Yukarıdaki programda aynı isim uzayı içerisinde **Sinif1** ve **Sinif2** sınıfları bulunmaktadır. **Sinif1** sınıfının **public** ve **statik** erişimli iki üyesi (**deger** değişkeni ve **Fonksiyon1** fonksiyonu) vardır. **Sinif1** sınıfı içinse herhangi bir erişim belirleyicisi belirtilmemiştir. Bu durumda **Sinif1** sınıfı, **internal** erişimli olacaktır yani, **Sinif1**' in tüm *public* üyeleri, aynı C#

programı (projesi) içerisinde yer alan tüm sınıflar tarafından erişilebilir olacaktır. Aynı proje içerisinde olmak kaydı ile, herhangi bir kod dosyasının içerisinde bulunan ve herhangi bir isim uzayında yer alan herhangi bir sınıf, **Uzay1** içerisindeki **Sinif1** sınıfına ulaşabilecektir. **Sinif2** sınıfı içerisinden **Sinif1**' in üyelerine ulaşmak da mümkün olacaktır ve **Main** fonksiyonunu sorunsuz bir biçimde çalışacaktır. Aşağıda verilen program da yukarıdakiyle birebir aynı anlamı taşımaktadır:

```
using System;

namespace Uzay1
{
    internal class Sinif1
    {
        public static int deger = 6;

        public static void Fonksiyon1()
        {
            Console.WriteLine("Sinif1 > Fonksiyon1");
        }
    }

    class Sinif2
    {
        static void Main()
        {
            Console.WriteLine("Sinif1 > deger : {0}", Sinif1.deger);
            Sinif1.Fonksiyon1();
            Console.ReadLine();
        }
    }
}
```

*Çıktı:*

```
Sinif1 > deger : 6
Sinif1 > Fonksiyon1
```

```
using System;

namespace Uzay1
{
    public class Sinif1
    {
        public static int deger = 6;

        public static void Fonksiyon1()
        {
            Console.WriteLine("Sinif1 > Fonksiyon1");
        }
    }

    class Sinif2
    {
        static void Main()
        {
            Console.WriteLine("Sinif1 > deger : {0}", Sinif1.deger);
            Sinif1.Fonksiyon1();
            Console.ReadLine();
        }
    }
}
```

*Çıktı:*

```
Sinif1 > deger : 6
Sinif1 > Fonksiyon1
```

Yukarıdaki programda ise **Sinif1** sınıfının erişim belirleyicisi **public** olarak seçilmiştir. Bu durumda ise **Sinif1**' in *public* erişimli üyelerine, aynı proje kapsamında olsun ya da olmasın tüm isim uzaylarında yer alan tüm sınıflardan erişilebilir.

Tıpkı isim uzayları gibi sınıflar da iç içe bulunabilirler. Bir sınıfın içerisinde yer alan sınıflara "**içsel sınıf**" adı verilir. **private** ve **protected** erişim belirleyicileri ise **sadece içsel sınıflarda kullanılabilirler**.

```
using System;

namespace Uzay1
{
    public class Sinif1
    { // erisim : public
        protected class Icsel1
        { // erisim : protected
            public static void Fonksiyon1()
            {
                Console.WriteLine("Sinif1.Icsel1 > Fonksiyon1");
            }
        }
    }

    class Sinif2
    { // erisim : internal
        static void Main()
        {
            Sinif1.Icsel1.Fonksiyon1(); // hata: erisim yok
            Console.ReadLine();
        }
    }
}
```

Yukarıdaki programda **Sinif1** içerisinde **Icsel1** isminde bir iç sınıf tanımlanmıştır. Bu içsel sınıfın erişimi **protected** olarak belirlendiği için, üyelerine **Sinif2** sınıfı içerisinden erişilmesi mümkün değildir, çünkü **Sinif2**, **Sinif1**' den miras alan bir sınıf değildir. Eğer **Icsel1** sınıfının erişim belirleyicisi **private** olsaydı, bu sınıfın üyelerine **Sinif2** içerisinden erişilmesi gene mümkün olmayacaktı. **Icsel1** sınıfı için herhangi bir erişim belirleyicisi yazılmaması da bu sınıfın **private** erişimli olması anlamına gelecektir.

```
using System;

namespace Uzay1
{
    public class Sinif1
    {
        protected class Icsel1
        { // erisim : protected
            public static void Fonksiyon1()
            {
                Console.WriteLine("Sinif1.Icsel1 > Fonksiyon1");
            }
        }
    }

    class Sinif2:Sinif1
    { // Sinif2 Sinif1' den miras alir
        static void Main()
        {
            Sinif1.Icsel1.Fonksiyon1();
            // YA DA: Icsel1.Fonksiyon1();
            Console.ReadLine();
        }
    }
}
```

Yukarıdaki programda ise **Sinif2 Sinif1**' den miras almaktadır. Böylelikle **Sinif1** içerisinde yer alan **protected** erişimli **Icsel1** içsel sınıfının *public* üyelerine **Sinif2** içerisinden erişilebilecektir. **Icsel1** sınıfının **internal** ya da **public** erişimli olması durumunda ise **Sinif2**, herhangi bir kalıtlama ilişkisine gereksinim olmaksızın **Icsel1** sınıfının *public* erişimli üyelerine erişebilecektir.

- **İçsel sınıflara, içinde bulundukları sınıfın nesneleri yaratılarak ulaşılamaz.** Aşağıdaki kod parçası hatalıdır:

```
...
Sinif1 nesne1 = new Sinif1();
nesne1.Icsel1.Fonksiyon1();
...
```

```
using System;

namespace Uzay1
{
    public class Sinif1
    { // public erisimli
        public int bolunen = 0;
        public int bolen = 0;

        public Sinif1(int s1, int s2)
        {
            bolunen = s1;
            bolen = s2;
        }

        public int Bol()
        {
            return (bolunen / bolen);
        }
    }

    class Sinif2
    { // internal erisimli
        static void Main()
        {
            Sinif1 nesne1 = new Sinif1(60, 15);
            int sonuc = nesne1.Bol();
            Console.WriteLine("Sonuc : {0}", sonuc);
            Console.ReadLine();
        }
    }
}
```

**Sinif1**' in erişim belirleyicisi **public** ise, bu sınıfın *public* erişimli olup da *statik olmayan* üyelerine **Sinif1** nesnesi yaratmak kaydı ile diğer tüm sınıflardan erişilebilir (yukarıdaki programda olduğu gibi). **Sinif1**' in **internal** erişimli olması durumunda ise sadece ilgili C# projesi kapsamındaki sınıflar tarafından nesneleri yaratılarak *public* erişimli üyelerine ulaşılabilecektir.

## Statik Sınıflar

```
using System;

namespace Uzay1
{
    public static class Sinif1
    { // public erisimli statik sinif

        // hata: statik siniflarin statik olmayan
        // uyeleri ve yapici metotlari bulunamaz
        public int bolunen = 0;
        public int bolen = 0;

        public Sinif1(int s1, int s2)
        {
            bolunen = s1;
            bolen = s2;
        }

        public int Bol()
        {
            return (bolunen / bolen);
        }
    }

    class Sinif2
    {
        static void Main()
        {
            Sinif1 nesne1 = new Sinif1(60, 15);
            int sonuc = nesne1.Bol();
            Console.WriteLine("Sonuc : {0}", sonuc);
            Console.ReadLine();
        }
    }
}
```

**statik** sınıfların ise statik olmayan üyesi (değişken, fonksiyon) bulunamaz ve bu sınıflarda yapıcı metot yer alamaz. Yukarıda yer alan program bu nedenle hatalıdır.

```
using System;

namespace Uzay1
{
    public static class Sinif1
    { // statik sinif
        public static int deger = 6;

        public static void Fonksiyon1()
        {
            Console.WriteLine("Sinif1 > Fonksiyon1");
        }
    }

    class Sinif2
    {
        static void Main()
        {
            Console.WriteLine("Sinif1 > deger : {0}", Sinif1.deger);
            Sinif1.Fonksiyon1();
            Console.ReadLine();
        }
    }
}
```

Yukarıdaki örnekte ise **Sinif1** sınıfı **public** erişimli **statik** bir sınıftır ve bu sınıfın tüm üyeleri de **statik**tir. Bu durumda, **Sinif2** içerisinden **Sinif1** sınıfının tüm üyelerine erişilebilmektedir. **Sinif1** sınıfının

```
internal static class Sinif1
```

```
static class Sinif1
```

yukarıdaki gibi **internal** erişimli olarak tanımlanması da **Sinif2** sınıfının aynı C# projesi içerisinde bulunması nedeniyle herhangi bir soruna yol açmayacaktır.

```
using System;

namespace Uzay1
{
    static class Sinif1
    { // statik sinif
        public static int deger = 6;

        public static void Fonksiyon1()
        {
            Console.WriteLine("Sinif1 > Fonksiyon1");
        }
    }

    class Sinif2
    {
        static void Main()
        {
            // statik bir sinifin nesnesi ve referansi olusturulamaz
            Sinif1 nesne1 = new Sinif1(); // hatali
            Console.ReadLine();
        }
    }
```

Statik sınıfların nesneleri yaratılamaz ve bu statik sınıfların nesnelerini refere etmek üzere sınıf tipinden referans üretilemez. Yukarıdaki program bu nedenle hatalıdır.

```
using System;

namespace Uzay1
{
    static class Sinif1
    { // statik sinif
        public static int deger = 6;

        public static void Fonksiyon1()
        {
            Console.WriteLine("Sinif1 > Fonksiyon1");
        }
    }

    class Sinif2:Sinif1
    { // hata: statik siniftan miras alinamaz
        static void Main()
        {
            Sinif1.Fonksiyon1(); // dogru kullanim
            Console.ReadLine();
        }
    }
}
```

Statik sınıflardan miras almak da mümkün değildir. Yukarıdaki program, **Sinif2 Sinif1**' den miras alacak şekilde tasarlanmıştır, hatalıdır.

```
using System;

namespace Uzay1
{
    static class Sinif1
    { // statik sinif
        public static int deger = 6;

        public static void Fonksiyon1()
        {
            Console.WriteLine("Sinif1 > Fonksiyon1");
        }
    }

    class Sinif2
    {
        static void Main()
        {
            Sinif1.Fonksiyon1(); // dogru kullanim
            Console.ReadLine();
        }
    }
}
```

Yukarıdaki program sorunsuz biçimde derlenecektir. Burada, **statik** bir sınıf olan **Sinif1**' in nesnesi yaratılmamış ve bu sınıftan miras alınmamıştır. **Sinif2** içerisinde yer alan **Main** fonksiyonunda, *statik* bir sınıf olan **Sinif1**' in içindeki *statik* bir fonksiyon olan **Fonksiyon1** çağrılmıştır. **Sinif1** sınıfı, "**internal static**" erişimli olduğu için ve **Sinif2** sınıfı ile aynı proje içerisinde bulunduğu için **fonksiyon1**' in çağrılması sorunsuz bir biçimde gerçekleşecektir.

## Soyut Sınıflar

Soyut sınıflar, **new** operatörü ile nesneleri yaratılamayan ancak kalıtlama yolu ile miras bırakabilen (kendisinden kalıtlanabilen) sınıflardır.

```
using System;

namespace Uzay1
{
    public abstract class Sinif1
    { // soyut sinif
        public int deger = 6;

        public void Fonksiyon1()
        {
            Console.WriteLine("Sinif1 > Fonksiyon1");
        }
    }

    class Sinif2
    {
        static void Main()
        { // hata: soyut siniflarin nesnesi yaratilamaz
            Sinif1 nesne1 = new Sinif1();
            nesne1.Fonksiyon1();
            Console.ReadLine();
        }
    }
}
```

Yukarıdaki program, **Sinif2** sınıfının içerisinde **soyut** bir sınıf olan **sinif1** sınıfının bir nesnesi yaratılmak istendiği için derlenmeyecektir. *Soyut* sınıflar, bu yönleri ile *statik* sınıflara benzemektedirler.

```
using System;

namespace Uzay1
{
    public abstract class Sinif1
    { // soyut sinif
        public static int deger = 6;

        public static void Fonksiyon1()
        {
            Console.WriteLine("Sinif1 > Fonksiyon1");
        }
    }

    class Sinif2:Sinif1
    { // soyut sınıflardan miras alinabilir
        static void Main()
        {
            Fonksiyon1();
            Console.ReadLine();
        }
    }
}
```

Soyut sınıflardan miras alınabilir. Soyut sınıflar bu yönleriyle *statik* sınıflardan ayrılırlar. Soyut sınıfların nesneleri yaratılamayacağı için, içerisinde bulunan **statik** üyelere erişim mümkün olacaktır. Yukarıdaki programda **Sinif2** sınıfı, *soyut* bir sınıf olan **Sinif1** sınıfından miras almaktadır. **Sinif1** içerisindeki *statik* yapıda olan **Fonksiyon1** fonksiyonu, miras yolu ile **Sinif2**' ye aktarılmıştır.

"Soyut sınıf" kavramı, "arayüz" konusu ile birlikte daha ayrıntılı olarak ele alınacaktır.

## Kapalı Sınıflar

Kapalı (**sealed**) sınıflar, kendisinden miras alınamayan sınıflardır.

```
using System;

namespace Uzay1
{
    public sealed class Sinif1
    { // kapali sinif
        public static int deger = 6;

        public static void Fonksiyon1()
        {
            Console.WriteLine("Sinif1 > Fonksiyon1");
        }
    }

    class Sinif2:Sinif1
    { // hata: kapali sınıflardan miras alinamaz
        static void Main()
        {
            Fonksiyon1();
            Console.ReadLine();
        }
    }
}
```

Yukarıdaki program derlenmeyecektir. **Sinif1** sınıfı, **kapalı** (**sealed**) bir sınıftır. Kapalı sınıfların miras bırakması mümkün değilken, **Sinif2**' nin **Sinif1**' den miras alması istenmektedir; bu hatalı bir gerçekleştirimdir.

```
using System;

namespace Uzay1
{
    public sealed class Sinif1
    { // kapali sinif
        public static int deger = 6;

        public static void Fonksiyon1()
        {
            Console.WriteLine("Sinif1 > Fonksiyon1");
        }
    }

    class Sinif2
    { // dogru kullanim
        static void Main()
        {
            Sinif1.Fonksiyon1();
            Console.ReadLine();
        }
    }
}
```

Kapalı sınıfların **public** erişimli *statik* üyelerine diğer sınıflardan nesne yaratmaksızın erişmek mümkündür.

```
using System;

namespace Uzay1
{
    public sealed class Sinif1
    { // kapali sinif
        public int deger = 6;

        public void Fonksiyon1()
        {
            Console.WriteLine("Sinif1 > Fonksiyon1");
        }
    }

    class Sinif2
    { // dogru kullanim
        static void Main()
        { // kapali siniflarin nesnesi yaratilabilir
            Sinif1 nesne1 = new Sinif1();
            nesne1.Fonksiyon1();
            Console.ReadLine();
        }
    }
}
```

Kapalı sınıfların nesnelerini yaratmak mümkündür. Yukarıdaki programda, **kapalı** bir sınıf olan **Sınıf1**, *statik* erişimli olmayan iki üyeye sahiptir. **Sinif2** içerisinde ise **Sinif1** sınıfının bir nesnesi yaratılmış ve **nesne1** referansı ile refere edilmiştir. Bu referans aracılığı ile, yaratılmış olan nesne üzerinden **Sinif1**' in **public** erişimli olan tüm üyelerine erişmek mümkün olacaktır.

- **Kapalı sınıflar, miras alan sınıf olabilirler.**

## Ek Bilgi: Dosya İşlemleri

```
using System;
using System.IO;

namespace DosyaIslemleri
{
    public class OkumaYazma
    {
        public static string Oku(string DosyaAdresi)
        { // dosyanin icerigini string olarak dondurur
            // DOSYA BULUNAMAZSA: CALISMA ANI HATASI
            string okunan = "";

            // StreamReader sinifi nesnesi yarat
            StreamReader sr = new StreamReader(DosyaAdresi);

            // nesneye mesaj gondererek ReadToEnd fonk. cagir
            okunan = sr.ReadToEnd();

            // okunan dosyayi serbest birakmak icin gerekli
            sr.Close();
            return okunan;
        }

        public static void Yaz(string DosyaAdresi, string metin)
        { // belli bir karakter dizisini dosyaya yazar

            // StreamWriter sinifi nesnesi yarat
            StreamWriter sw = new StreamWriter(DosyaAdresi);

            // ilgili dosyaya metin in icerigini yaz
            sw.Write(metin);

            // kaydet ve dosyayi serbest birak
            sw.Close();
        }
    }

    class Program
    {
        static void Main()
        {
            string pascal = "1\n1\t1\n1\t2\t1\n1\t3\t3\t1\n1\t4\t6\t4\t1";

            // WINDOWS: "C:\" altinda "kaynak.txt" dosyasi olmali
            // UNIX : bir metin dosyasi olusturarak adresini giriniz
            // okunan dosyanin icerigi str icerisine aktarildi
            string str = OkumaYazma.Oku("C:/kaynak.txt");

            // WINDOWS : "C:\" altinda "hedef.txt" dosyasi olusturulacak
            // UNIX : istediginiz dizinde istediginiz dosya adini giriniz
            // pascal in icerigi hedef dosyaya yazilacak
            OkumaYazma.Yaz("C:/hedef.txt", pascal);

            // dosyadan okunan metni ekrana yaz
            Console.WriteLine(str);
            Console.ReadLine();
        }
    }
```

Yukarıda, en basit şekliyle sabit diskte bulunan bir metin dosyasında yer alan karakterleri okuyarak karakter dizisi şeklinde döndüren ve belli bir karakter dizisini, disk üzerinde belirlenen bir dizinde oluşturulan dosya içerisine kaydeden bir program verilmiştir.

İlgili programda verilen **OkumaYazma** sınıfı içerisinde "**Oku**" ve "**Yaz**" isimlerinde iki farklı fonksiyon yer almaktadır.

**Oku** fonksiyonu, *string* tipinde tek bir argüman almakta ve *string* tipinde bir değer döndürmektedir. Bu fonksiyon, argüman olarak okunacak olan dosyanın sabit disk üzerindeki adresini (tam adres ya da bağıl adres) almaktadır. **Oku** fonksiyonu çağrıldığında, dosyadan okuma işlemlerinin yer aldığı **StreamReader** sınıfının bir nesnesini yaratmaktadır. Nesnenin yaratılması sırasında, bu sınıfın tek argüman alan yapıcı metodu çağrılmış ve bu metoda argüman olarak, okunacak olan dosyanın diskteki adresini tutan *string* tipindeki bir değişken (**Oku** fonksiyonunun almış olduğu argüman) verilmiştir. Daha sonra, yaratılmış olan nesne aracılığı ile **StreamReader** sınıfının statik olmayan bir fonksiyonu olan "**ReadToEnd**" fonksiyonu çağrılmıştır. Argüman almayan bu fonksiyon, değer olarak yapıcı metot aracılığı ile adresi aktarılan dosyanın baştan sona tüm içeriğini (yeni satır karakterleri de dâhil olmak üzere), karakter dizisi olarak (*string* tipinde) döndürmektedir. **Oku** fonksiyonu içerisinde yer alan ve *string* tipindeki **okunan** değişkeni ile, **ReadToEnd** fonksiyonunun dönüş değeri olan dosya içeriği refere edilmektedir. Bu değişken, aynı zamanda **Oku** fonksiyonunun dönüş değeridir. **Oku** fonksiyonu sonlanmadan, yaratılmış olan **StreamReader** nesnesi aracılığı ile bu sınıfın statik olmayan bir diğer fonksiyonu olan **Close** fonksiyonu çağrılmıştır. Buradaki amaç, kullanılmak üzere işletim sistemi tarafından elimizdeki programa tahsis edilmiş olan dosyanın serbest bırakılması ile diğer uygulamalar tarafından da kullanılabilmesini mümkün kılmaktır. **StreamReader** sınıfı, **System** isim uzayı içerisinde bulunan **IO** alt isim uzayında yer almaktadır.

**Yaz** fonksiyonu ise, *string* tipinde iki argüman almakta ve değer döndürmemektedir. Argüman olarak alınan değerler, içerisine yazılacak olan dosyanın diskteki adresi ve bu dosyaya yazılacak olan karakter dizisidir. **Yaz** fonksiyonunun çağrılmasıyla, dosyaya yazma işlemlerinin gerçekleştirilmesi için tasarlanmış olan **StreamWriter** sınıfının bir nesnesi yaratılmaktadır. Nesne yaratılırken, **StreamWriter** sınıfının tek argüman alan yapıcı metodu çağrılmış ve argüman olarak, içerisine yazma işlemi yapılacak olan dosyanın adresi

verilmiştir. Daha sonra, yaratılan nesne üzerinden **StreamWriter** sınıfının statik olmayan bir fonksiyonu olan **Write** fonksiyonu çağrılmıştır. Bu fonksiyon, argüman olarak, yapıcı metoda adresi argüman olarak verilmiş dosyaya yazılacak olan karakter dizisini almakta ve bu diziyi ilgili dosyanın içerisine yazmaktadır (Adresi verilen dosya mevcut değilse yeniden yaratılmaktadır.). Sonrasında ise, aynı sınıfın **Close** fonksiyonu çağrılarak içerisine yazma işlemi yapılan dosyanın yeni içeriği kaydedilmekte ve dosya serbest bırakılmaktadır.