

MİRAS (KALITIM)

Miras Kavramı

C# programlama dilinde sınıflar, başka bir sınıftan ya da arayüz(ler)den miras alabilirler (kalıtlayabilirler).

Bir Sınıfın Diğer Bir Sınıftan Miras Alması

Miras alan bir sınıf, miras aldığı sınıfın **mirasla aktarılabılır nitelikteki** tüm üyelerini bünyesinde bulundurmuş olur. Miras veren bir sınıfın herhangi bir üyesinin (değişken, fonksiyon) bu sınıftan miras alan diğer bir sınıfa aktarılabilmesi için, ilgili üyenin erişim belirleyicisinin **public**, **internal** ya da **protected** olması gerekir. **private** erişim belirleyicisine sahip sınıf üyeleri, miras yolu ile aktarılamazlar. Miras veren sınıftaki bir üyenin **protected** erişimli olması, bu üyenin, içerisinde bulunmakta olduğu sınıftan miras alan diğer sınıflara aktarılacağı ancak aynı sınıftan miras almayan sınıflar için erişilemez olacağı anlamını taşımaktadır.

```
using System;

public class Sinif1
{
    public int a, b;

    public int topla()
    {
        return (a + b);
    }

    public static void argumaniYaz(string arguman)
    {
        Console.WriteLine(arguman);
    }

    public Sinif1()
    {
        a = 0;
        b = 0;
    }

    public Sinif1(int arg1, int arg2)
    {
        a = arg1;
        b = arg2;
    }
}

public class Sinif2 : Sinif1
{
    public Sinif2(int arg1, int arg2)
    {
        a = arg1 + 1;
        b = arg2 + 1;
    }
}
```

Çıktı:

```
Sinif3' ten merhaba...
Toplam : 9
nesne1.a : 4, nesne1.b : 5
```

```

public class Sinif3
{
    static void Main()
    {
        Sinif2 nesne1 = new Sinif2(3, 4);
        int toplam = nesne1.topla();
        Sinif2.argumaniYaz("Sinif3' ten merhaba...");
        Console.WriteLine("Toplam : {0}", toplam);
        Console.WriteLine("nesne1.a : {0}, nesne1.b : {1}", nesne1.a, nesne1.b);

        // Sinif2 nesne2 = new Sinif2();
        // hata: Sinif2' de varsayılan yapıcı metot tanımlanmamış
        Console.ReadLine();
    }
}

```

(devam)

Yukarıdaki örnekte verilen **Sinif1** sınıfında üye olarak 2 değişken (**a** ve **b**), 2 metot (**topla** ve **argumaniYaz**) ve 2 yapıcı metot (argüman almayan ve 2 argüman alan) yer almaktadır. Bu üyelerin erişim belirleyicilerinin tamamı **public** olduğundan, bunların (yapıcı metotlar hariç) kalıtılma yolu ile miras alan sınıfa aktarılmasında herhangi bir engel bulunmamaktadır. Ancak her ne kadar **public** erişim belirleyicisine sahip olsalar da, bir sınıfın yapıcı metotları, kalıtılma yolu ile miras alan sınıfa aktarılmazlar; miras alan sınıfın, kendine ait yapıcı metotları gerçeklemesi gerekmektedir. Verilen örnekte **Sinif2** sınıfı, **Sinif1** sınıfından miras almaktadır. **Sinif2** içerisinde, sadece 2 argüman alan yapıcı metotun gerçekleştirilmesi yapılmıştır (Eğer **Sinif1** içerisinde argüman almayan yapıcı metodun gerçekleştirilmesi yapılmasaydı, **Sinif2** içerisinde argüman almayan bir yapıcı metot gerçekleştirilmesi zorunlu olacaktı.). **Sinif3** sınıfı içerisinde yer alan **Main** fonksiyonu içinde ise **Sinif2** sınıfının **nesne1** isminde bir nesnesi yaratılmıştır. Bu nesnenin yaratılması esnasında çağrılan yapıcı metot, **Sinif2** içerisinde tanımlanmış olan ve 2 argüman alan yapıcı metottur. Sonraki adımlarda **nesne1** üzerinden çağrılan **topla** fonksiyonu ile **Sinif2**' nin ismi belirtilerek çağrılan ve statik olmayan **argumaniYaz** fonksiyonu ise, **Sinif2** içerisinde açıkça gerçekleştirilmedikleri halde **Sinif2**' ye **Sinif1**' den miras yolu ile aktarılmış fonksiyonlardır. **Sinif1** sınıfının değişken üyeleri olan **a** ve **b** de, **Sinif2** sınıfına miras yolu ile aktarıldıkları için **Sinif2** nesnelerinin alt alanları olarak kullanılabilirler.

```

using System;

public class Sinif1
{
    protected int a, b;

    ...
}

```

Verilen örnek programda **Sinif1** içerisinde yer alan **a** ve **b** değişken üyelerinin erişim belirleyicisi **protected** yapıldığında, **Sinif3** içerisinde yaratılan **nesne1** nesnesinin **a** ve **b** alt

alanlarına erişim olanaksız hale geleceğinden derleme anı hatası ile karşılaşılacaktır.

```
using System;

public class Sinif1
{
    private int a, b;

    ...
}
```

Benzer şekilde, **Sinif1** içerisinde yer alan **a** ve **b** değişken üyelerinin erişim belirleyicisi **private** yapıldığında; gerek **Sinif2'** nin yapıcı metodunda, aslında **Sinif2'** ye kalıtımla aktarılmamış olan **a** ve **b** üyelerine değer ataması yapılmak istendiği için, gerekse **Sinif3** içerisinde yaratılan **nesne1** nesnesinin **a** ve **b** alt alanlarına erişim olanaksız hale geldiği için derleme anı hatası ile karşılaşılacaktır.

```
using System;

public class Sinif1
{
    public int a;

    public int islemYap()
    {
        return (a * a);
    }
    public Sinif1()
    {
        a = 0;
    }
    public Sinif1(int arg1)
    {
        a = arg1;
    }
}

public class Sinif2 : Sinif1
{
    public Sinif2(int arg1)
    {
        a = arg1;
    }
}

public class Sinif3
{
    static void Main()
    {
        Sinif2 nesne1 = new Sinif2(3);
        int sonuc = nesne1.islemYap();
        Console.WriteLine("İslem Sonucu : {0}", sonuc);
        Console.ReadLine();
    }
}
```

Çıktı:

İslem Sonucu : 9

Yukarıdaki program incelendiğinde, **Sinif2** sınıfının **Sinif1'** den miras aldığı görülmektedir. Doğal olarak, **Sinif1'** de yer alan **islemYap** fonksiyonu da miras yolu ile **Sinif2'** ye aktarılmıştır. **Sinif3** sınıfı içerisindeki **Main** fonksiyonunda da **Sinif2** sınıfının bir nesnesi yaratılmış ve daha sonra bu nesne üzerinden **islemYap** fonksiyonu çağırılmıştır. Bu çağrı, **Sinif1** sınıfı içerisinde gerçekleşmiş olan **islemYap** fonksiyonunun miras yolu ile **Sinif2**

içerisine aktarılmış olan kopyasının çağırılması işlemidir. Şimdi, **Sinif2** sınıfını aşağıdaki biçimde güncelleyelim:

<pre>... public class Sinif2 : Sinif1 { public int islemYap() { return (a * a * a); } public Sinif2(int arg1) { a = arg1; } } ...</pre>	<p>Çıktı:</p> <p>İşlem Sonucu : 27</p>
--	--

Bu güncellemeden sonra **Sinif3** içindeki **Main** fonksiyonunda **Sinif2'** nin bir nesnesinin yaratılması ve bu nesne üzerinden **islemYap** fonksiyonunun çağırılması durumunda çağrılan fonksiyon, **Sinif2'** ye sonradan eklenmiş olan ve sınıf içerisindeki **a** değişken üyesinin küpünü döndüren fonksiyon olacaktır. Bu durumda, **Sinif1** sınıfından **Sinif2** sınıfına miras yolu ile aktarılmış olan ve sınıf içindeki **a** değişken üyesinin karesini döndüren fonksiyon **geçersiz kılınmış** (*geçersiz kılmak: override*) olacaktır ve bu fonksiyonun yerini, aynı isme sahip olan ancak sınıf içindeki **a** değişken üyesinin küpünü döndüren, **Sinif2** içerisinde sonradan tanımlanmış olan **islemYap** fonksiyonu alacaktır.

Böylesi bir durumda derleyici, **Sinif2** içerisinde tanımlanan **islemYap** fonksiyonuna yanlışıyla, mirasla alınan **islemYap** fonksiyonununkiyle aynı adın verilmesinden hareketle uyarı mesajı verebilir. Bu uyarı mesajının önüne geçmek, yani **geçersiz kılma işlemini bilinçli olarak yaptığımızı derleyiciye ifade etmek** için, **new** anahtar sözcüğünden şu şekilde yararlanabiliriz:

<pre>... public class Sinif2 : Sinif1 { new public int islemYap() { return (a * a * a); } public Sinif2(int arg1) { a = arg1; } } ...</pre>
--

Geçersiz kılma işlemi, fonksiyonlarda olduğu gibi sınıf üyesi değişkenlerde de söz konusudur:

```
...
public class Sinif2 : Sinif1
{
    new public int a;
    ...
}
```

```
using System;

public class Sinif1
{
    public int a;

    public int islemYap1()
    {
        return (a * a);
    }
    public Sinif1()
    {
        a = 0;
    }
    public Sinif1(int arg1)
    {
        a = arg1;
    }
}

public class Sinif2 : Sinif1
{
    public int islemYap2()
    {
        return (a * a * a);
    }

    public Sinif2(int arg1)
    {
        a = arg1;
    }
}

public class Sinif3
{
    static void Main()
    {
        Sinif2 nesne1 = new Sinif2(3);
        int sonuc1 = nesne1.islemYap1();
        int sonuc2 = nesne1.islemYap2();
        Console.WriteLine("1. Islem Sonucu : {0}", sonuc1);
        Console.WriteLine("2. Islem Sonucu : {0}", sonuc2);

        Console.ReadLine();
    }
}
```

Çıktı:

```
1. Islem Sonucu : 9
2. Islem Sonucu : 27
```

Miras alan bir sınıf için; mirasla gelen ve bu sınıf içerisinde yeniden gerçekleştirilmek suretiyle geçersiz kılınmayan bir fonksiyon (örnekteki **islemYap1** fonksiyonu) ile, yalnızca bu sınıf içerisinde gerçekleştirilen bir fonksiyon (örnekteki **islemYap2** fonksiyonu) aynı konumda olup, her ikisi de bu sınıfın birer üyesi olarak işlem görürler.

statik erişimli fonksiyonlar söz konusu olduğunda da "kalıtım yoluyla aktarım" ve "geçersiz kılma" kavramları geçerli olmaktadır. Aşağıdaki örneği inceleyelim:

```

using System;

public class Sinif1
{
    public static int islemYap(int arg1)
    {
        return (arg1 * arg1);
    }
}

public class Sinif2 : Sinif1
{
}

public class Sinif3
{
    static void Main()
    {
        int sonuc1 = Sinif1.islemYap(5);
        int sonuc2 = Sinif2.islemYap(5);
        Console.WriteLine("1. Islem Sonucu : {0}", sonuc1);
        Console.WriteLine("2. Islem Sonucu : {0}", sonuc2);
        Console.ReadLine();
    }
}

```

Çıktı:

```

1. Islem Sonucu : 25
2. Islem Sonucu : 25

```

Yukarıda verilen örnek programda, **Sinif2'** nin **Sinif1'** den miras aldığı görülmektedir. Bu durumda, **Sinif1'** in tek üyesi olan ve **statik** erişimli **islemYap** fonksiyonunun bir kopyası, kalıtım yolu ile **Sinif2'** ye aktarılabacaktır. **Sinif3'** te yer alan **Main** fonksiyonu içerisinde hem **Sinif1'** e ait olan **islemYap** fonksiyonu, hem de **Sinif2'** ye ait olan **islemYap** fonksiyonu çağırılmış ve her iki fonksiyon çağırısından da aynı sonuç elde edilmiştir.

```

...

public class Sinif2 : Sinif1
{
    new public static int islemYap(int arg1)
    {
        return (arg1 * arg1 * arg1);
    }
}
...

```

Çıktı:

```

1. Islem Sonucu : 25
2. Islem Sonucu : 125

```

Sinif2 sınıfı yukarıdaki gibi yeniden düzenlenirse, **Sinif2** içerisine miras yolu ile aktarılmış olan ve argüman olarak aldığı sayının karesini döndüren **islemYap** fonksiyonu geçersiz kılınmış olacaktır ve bu fonksiyonun yerini, **Sinif2** içerisinde sonradan tanımlanmış olan ve argüman olarak aldığı sayının küpünü döndüren **islemYap** fonksiyonu alacaktır.

```
using System;

public class Sinif1
{
    public static int islemYap(int arg1)
    {
        return (arg1 * arg1);
    }
}

public class Sinif2 : Sinif1
{
    new private static int islemYap(int arg1)
    { // gecersiz kilma soz konusu degil
        return (arg1 + arg1);
    }
}

public class Sinif3 : Sinif2
{
}

public class Sinif4
{
    static void Main()
    {
        int sonuc = Sinif3.islemYap(5);
        Console.WriteLine("Islem Sonucu : {0}", sonuc);
        Console.ReadLine();
    }
}
```

Çıktı:

Islem Sonucu : 25

Sınıf dışından bakıldığında, bir sınıf içerisinde tanımlanan **private** erişimli bir fonksiyon, bu sınıfa miras yoluyla gelen ve aynı ismi taşıyan fonksiyonu geçersiz kılmaz. **Sinif4** içerisindeki **Main** fonksiyonunda, **Sinif3** sınıfına ait **islemYap** fonksiyonu çağrıldığında, **Sinif2'** ye **Sinif1'** den miras yoluyla geçen ve argüman olarak aldığı sayının karesini döndüren **islemYap** fonksiyonu çağrılmış olacaktır.

```
using System;

public class Sinif1
{
    public static int islemYap(int arg1)
    {
        return (arg1 * arg1);
    }
}

public class Sinif2 : Sinif1
{
    new private static int islemYap(int arg1)
    {
        return (arg1 + arg1);
    }
    public static void fonk(int arg)
    {
        Console.WriteLine("Islem Sonucu : {0}", islemYap(arg));
    }
}

public class Sinif3 : Sinif2
{
}

public class Sinif4
{
    static void Main()
    {
        Sinif3.fonk(5);
        Console.ReadLine();
    }
}
```

Çıktı:

Islem Sonucu : 10

Sınıf içinden bakıldığında ise, bir sınıf içerisinde tanımlanan **private** erişimli bir fonksiyon, bu sınıfa miras yoluyla gelen ve aynı ismi taşıyan fonksiyonu geçersiz kılar. Yukarıdaki örnekte; **Sinif2** sınıfı içerisinde tanımlanmış olan, **private** erişimli olan ve argüman olarak aldığı sayının iki katını döndüren **islemYap** fonksiyonu, gene **Sinif2** sınıfı içinde bulunan **fonk** fonksiyonu içerisinde yapılan "**islemYap**" isimli fonksiyon çağrısında, **Sinif2**'ye **Sinif1**'den miras yoluyla geçen ve argüman olarak aldığı sayının karesini döndüren **islemYap** fonksiyonunu geçersiz kılmıştır.

```
using System;

public class Sinif1
{
    public int a = 7;

    public int islemYap()
    {
        return (a * a);
    }
}

public class Sinif2 : Sinif1
{
    public static int islemYap()
    { // gecersiz kılma söz konusu
        return 500;
    }
}

public class Sinif3 : Sinif2
{
}

public class Sinif4
{
    static void Main()
    {
        Sinif3 nesne1 = new Sinif3();
        Console.WriteLine("1. Islem Sonucu : {0}", nesne1.islemYap()); // hata
        Console.WriteLine("2. Islem Sonucu : {0}", Sinif3.islemYap());
        Console.ReadLine();
    }
}
```

Adları ve argüman alma biçimleri aynı olan fonksiyonlar arasında geçersiz kılma ilişkisi söz konusudur. Bu fonksiyonlardan birinin **statik** erişimli olup diğerinin **statik** erişimli olmaması geçersiz kılma durumunu ortadan kaldırmaz. Yukarıdaki örnekte, **Sinif2** içerisinde gerçekleşen ve **statik** erişimli olan **islemYap** fonksiyonu, **Sinif1**'den **Sinif2**'ye miras yolu ile geçen ve statik erişimli olmayan **islemYap** fonksiyonunu geçersiz kılmıştır. Dolayısı ile; **Sinif4** içinde bulunan **Main** fonksiyonunda **islemYap** fonksiyonunun **Sinif3** nesnesi üzerinden çağırılması, derleme anı hatasına yol açacaktır.

Yukarıdaki örnekte, **Sinif2** içerisinde yer alan **islemYap** fonksiyonunun argüman alma biçiminde değişiklik yapılırsa, fonksiyon adlarının aynı olmasına rağmen geçersiz kılma durumu söz konusu olmaz:


```

using System;

public class Sinif1
{
    public int a = 7;
    public int islemYap()
    {
        return (a * a);
    }
}

public class Sinif2 : Sinif1
{
    public static int islemYap(int b)
    { // gecersiz kilma soz konusu degil; adlar ayni, argumanlar farkli
        return b;
    }
}

public class Sinif3 : Sinif2
{
}

public class Sinif4
{
    static void Main()
    {
        Sinif3 nesne1 = new Sinif3();
        Console.WriteLine("1. Islem Sonucu : {0}", nesne1.islemYap()); // #1
        Console.WriteLine("2. Islem Sonucu : {0}", Sinif3.islemYap(9)); // #2
        Console.ReadLine();
    }
}

```

Yukarıdaki programda; **Sinif4** içerisinde yer alan **Main** fonksiyonu içinde **#1** ile işaretlenmiş fonksiyon çağrısına cevap veren fonksiyon, **Sinif1**' den **Sinif2**' ye mirasla aktarılan ve statik erişimli olmayan **islemYap** fonksiyonu iken; **#2** ile işaretlenmiş statik fonksiyon çağrısına cevap veren fonksiyon, **Sinif2** içerisinde gerçekleştirilen ve **statik** erişimli olan (aynı zamanda 1 argüman alan) **islemYap** fonksiyonu olmaktadır.

C# programlama dilinde bir sınıfın birden çok sayıda sınıftan miras alması mümkün değildir:

```

using System;

public class Sinif1
{
    public static int islemYap1(int arg1)
    {
        return (arg1 * arg1);
    }
}

public class Sinif2
{
    public static int islemYap2(int arg1)
    {
        return (arg1 + arg1);
    }
}

public class Sinif3 : Sinif1, Sinif2
{ // hata: birden fazla sayida siniftan miras alinamaz
}

public class Sinif4
{
    static void Main()
    {
        int sonuc1 = Sinif3.islemYap1(5);
        int sonuc2 = Sinif3.islemYap2(5);
        Console.WriteLine("1. Islem Sonucu : {0}", sonuc1);
        Console.WriteLine("2. Islem Sonucu : {0}", sonuc2);
        Console.ReadLine();
    }
}

```

Yukarıdaki program, **Sinif3** sınıfının birden fazla sınıftan miras alması istendiği için (Bir sınıfın birden fazla kaynaktan miras aldığı durumlarda bu kaynaklar virgüllerle ayrılır. Burada sözdizimsel bir hata yoktur, mantıksal hata vardır.) hatalıdır.

Ancak, mirasın kuşaktan kuşağa aktarılması mümkündür:

```
using System;

public class Sinif1
{ // uyeler: islemYap1
    public static int islemYap1(int arg1)
    {
        return (arg1 * arg1);
    }
}

public class Sinif2 : Sinif1
{ // uyeler: islemYap1, islemYap2
    public static int islemYap2(int arg1)
    {
        return (arg1 + arg1);
    }
}

public class Sinif3 : Sinif2
{ // uyeler: islemYap1, islemYap2
}

public class Sinif4
{
    static void Main()
    {
        int sonuc1 = Sinif3.islemYap1(5);
        int sonuc2 = Sinif3.islemYap2(5);
        Console.WriteLine("1. Islem Sonucu : {0}", sonuc1);
        Console.WriteLine("2. Islem Sonucu : {0}", sonuc2);
        Console.ReadLine();
    }
}
```

Çıktı:

```
1. Islem Sonucu : 25
2. Islem Sonucu : 10
```

Yukarıdaki programda, **Sinif1** sınıfının **islemYap1** isminde tek bir üyesi (fonksiyon) bulunmaktadır. **Sinif2** sınıfı içerisinde üye olarak **islemYap2** fonksiyonu tanımlanmış; bu üyenin yanı sıra, **Sinif1** sınıfından miras alındığı için, **Sinif1**'de gerçekleşen **islemYap1** fonksiyonunun bir kopyası da **Sinif2** sınıfı içerisinde bir diğer üye olarak yer almıştır. İçerisinde herhangi bir üye tanımlanmayan **Sinif3** fonksiyonu ise, **islemYap2** ve **islemYap1** (kalıtılma yolu ile) üyelerini bünyesinde bulunduran **Sinif2** sınıfından miras aldığından, **islemYap1** ve **islemYap2** fonksiyonlarını üye olarak bulundurmaktadır.

```
using System;
// hata : dongusel bir kalitlama iliskisi kurulamaz
public class Sinif1 : Sinif3
{
    public static int islemYap1(int arg1)
    {
        return (arg1 * arg1);
    }
}
public class Sinif2 : Sinif1
{
    public static int islemYap2(int arg1)
    {
        return (arg1 + arg1);
    }
}
public class Sinif3 : Sinif2
{
}
public class Sinif4
{
    static void Main()
    {
        int sonuc1 = Sinif3.islemYap1(5);
        int sonuc2 = Sinif3.islemYap2(5);
        Console.WriteLine("1. Islem Sonucu : {0}", sonuc1);
        Console.WriteLine("2. Islem Sonucu : {0}", sonuc2);
        Console.ReadLine();
    }
}
```

Yukarıdaki program ise, **kalıtlama ilişkisinin döngüsel bir biçimde kurulmasının mümkün olmaması** nedeniyle hatalıdır. Verilen programda; **Sinif2** sınıfının **Sinif1** sınıfından, **Sinif3** sınıfının **Sinif2** sınıfından, **Sinif1** sınıfının ise **Sinif3** sınıfından miras alması istendiği için derleme anı hatası ile karşılaşılması söz konusu olmaktadır.

Bir Sınıfın Bir ya da Birden Fazla Arayüzden Miras Alması

C# programlama dilinde bir sınıfın miras alabileceği sınıf sayısının en fazla bir olduğu, bir sınıfın birden çok sayıda kaynaktan miras alabilmesini mümkün kılmak adına sınıfların bir ya da birden çok arayüzden miras almasının mümkün kılındığı daha önceki konularda ele alınmıştı. Ayrıca bir arayüzün, bir ya da birden çok sayıda arayüzden miras almasının mümkün olduğuna da değinilmişti.

Bir ya da birden fazla sayıda arayüzden miras alan bir sınıf, bu arayüzlerin tümünde yer alan üyelerin (fonksiyon imzası, özellik) tamamını gerçeklemek zorundadır. Arayüz üyeleri **statik** olamazlar. Aşağıdaki örnek programı hatırlayalım:

```
using System;

namespace Uzay1
{
    public interface ISinif1
    { // ISinif1 arayuzu
        int ozellik1 { get; set; }
        int fonksiyon1();
    }
    public interface ISinif2
    { // ISinif2 arayuzu
        string ozellik2 { get; set; }
        string fonksiyon2();
    }
    public interface ISinif3 : ISinif1, ISinif2
    { // ISinif1 ve ISinif2' den miras alan ISinif3 arayuzu
        string fonksiyon3();
    }
    public class Sinif1 : ISinif3
    { // ISinif3 arayuzunden kalıtlayan sınıf
        public int i1;
        public string s1;

        public Sinif1()
        {
            i1 = 9;
            s1 = "dokuz";
        }
        public int ozellik1
        {
            get { return i1; }
            set { i1 = value; }
        }
        public string ozellik2
        {
            get { return s1; }
            set { s1 = value; }
        }
        public int fonksiyon1()
        {
            return 5;
        }
        public string fonksiyon2()
        {
            return "bes";
        }
    }
}
```

Çıktı:

```
9
77
dokuz
yetmişyedi
nesnel.fonksiyon1() --> 5
nesnel.fonksiyon2() --> bes
nesnel.fonksiyon3() --> bir
```

```

        public string fonksiyon3()
        {
            return "bir";
        }
    }
    class Sinif2
    {
        static void Main()
        {
            Sinif1 nesne1 = new Sinif1();
            Console.WriteLine("{0}", nesne1.ozellik1);
            nesne1.ozellik1 = 77;
            Console.WriteLine("{0}", nesne1.ozellik1);

            Console.WriteLine("{0}", nesne1.ozellik2);
            nesne1.s1 = "yetmişyedi";
            Console.WriteLine("{0}", nesne1.ozellik2);

            Console.WriteLine("nesne1.fonksiyon1() --> {0}", nesne1.fonksiyon1());
            Console.WriteLine("nesne1.fonksiyon2() --> {0}", nesne1.fonksiyon2());
            Console.WriteLine("nesne1.fonksiyon3() --> {0}", nesne1.fonksiyon3());
            Console.ReadLine();
        }
    }
}

```

(devam)

Yukarıdaki örnekte **ISinif1** ve **ISinif2**, herhangi bir kaynaktan miras almayan başlı başına iki ayrı arayüzdür. **ISinif3** arayüzü ise hem **ISinif1**, hem de **ISinif2** arayüzünden miras almaktadır. Bu durumda **ISinif3** arayüzünden miras alan tüm sınıflar; hem **ISinif1** arayüzünün üyelerini, hem **ISinif2** arayüzünün üyelerini, hem de **ISinif1**' de ve **ISinif2**' de yer almayan ancak **ISinif3**' te bulunan üyeleri gerçeklemek zorundadırlar. Verilen örnekte **ISinif1** arayüzü içerisinde **ozellik1** özelliğine ve **fonksiyon1** fonksiyonuna ait imzalar (prototipler), **ISinif2** arayüzü içerisinde **ozellik2** özelliğine ve **fonksiyon2** fonksiyonuna ait imzalar, **ISinif3** arayüzü içerisinde ise **fonksiyon3** fonksiyonuna ait imza yer almaktadır. **Sinif1** sınıfı ise **ISinif3** arayüzünden miras almaktadır. Bu miras ilişkisinin sonucu olarak **Sinif1** sınıfı; **ozellik1** ve **ozellik2** özellikleri ile **fonksiyon1**, **fonksiyon2** ve **fonksiyon3** fonksiyonlarını gerçeklemek zorundadır.

```

using System;

namespace Uzay1
{
    public interface ISinif1
    { // ISinif1 arayuzu
        int fonksiyon1(int a);
    }
    public interface ISinif2
    { // ISinif2 arayuzu
        int fonksiyon1(int b);
    }
    public interface ISinif3 : ISinif1, ISinif2
    { // ISinif1 ve ISinif2' den miras alan ISinif3 arayuzu
    }
}

```

Çıktı:

Sonuc : ?

```

public class Sinif1 : ISinif3
{ // ISinif3 arayuzunden miras alan sinif
    public int i1 = 6;

    public int fonksiyon1(int x)
    {
        return x;
    }
}
class Sinif2
{
    static void Main()
    {
        Sinif1 nesne1 = new Sinif1();
        int sonuc = nesne1.fonksiyon1(7);
        Console.WriteLine("Sonuc : {0}", sonuc);
        Console.ReadLine();
    }
}
}

```

(devam)

Yukarıdaki örnekte, **ISinif1** ve **ISinif2** arayüzlerinde, adları ve imzaları (prototipleri) birbirinin tamamen aynısı olan (Argümanların adlandırılması farklı olabilir.) **fonksiyon1** isimli üyeler bulunmaktadır. **ISinif3** arayüzünün hem **ISinif1**' den, hem de **ISinif2**' den miras alması söz konusudur. Miras veren sınıfların her ikisinin de mirasla aktaracakları üyenin birebir aynı olması (**fonksiyon1** isminde, **int** tipinde tek bir argüman alan ve **int** tipinde değer döndüren fonksiyon imzası) bir probleme yol açmayacaktır. Böylesi bir durumda, **ISinif3** arayüzünde üye olarak **fonksiyon1** isminde, **int** tipinde tek bir argüman alan ve **int** tipinde değer döndüren yalnızca bir tane fonksiyon imzası bulunacaktır.

ISinif2 arayüzünde şu değişiklikler yapılsın:

```

public interface ISinif2
{ // ISinif2 arayuzu
    int fonksiyon1(long b);
}

```

I

```

public interface ISinif2
{ // ISinif2 arayuzu
    long fonksiyon1(int b);
}

```

II

```

public interface ISinif2
{ // ISinif2 arayuzu
    int fonksiyon2(int b);
}

```

III

- I** ile belirtilen değişiklik yapıldığında **ISinif3** arayüzüne; **ISinif1**' den **fonksiyon1** adında, **int** tipinden tek bir argüman alan ve **int** tipinden değer döndüren bir üye ile **ISinif2**' den **fonksiyon1** adında, **long** tipinden tek bir argüman alan ve **int** tipinden

değer döndüren bir üye olmak üzere toplamda iki farklı üye aktarılacaktır. **ISinif3** arayüzünden miras alan **Sinif1** sınıfı ise imzası verilen bu iki üyeyi de gerçeklemek zorunda kalacaktır. Yani, **Sinif1** içerisinde aynı ada ve aynı dönüş değerine sahip ancak argüman tipleri farklı iki fonksiyon yer alacaktır. Her ne kadar bu fonksiyonların isimleri ve dönüş değer tipleri aynı olsa da, argüman tipleri farklı olduğundan fonksiyon çağrısı yapıldığında hangi fonksiyonun görev alacağı, fonksiyona argüman olarak verilen değer tipine göre belirlenecektir:

```
using System;

namespace Uzay1
{
    public interface ISinif1
    { // ISinif1 arayuzu
        int fonksiyon1(int a);
    }
    public interface ISinif2
    { // ISinif2 arayuzu
        int fonksiyon1(long b);
    }
    public interface ISinif3 : ISinif1, ISinif2
    { // ISinif1 ve ISinif2' den miras alan ISinif3 arayuzu
    }
    public class Sinif1 : ISinif3
    { // ISinif3 arayuzundan miras alan sınıf
        public int i1 = 6;

        public int fonksiyon1(int x)
        {
            Console.WriteLine("CAGRILAN FONKSIYON : [ int fonksiyon1(int a); ]");
            return (x + 1);
        }
        public int fonksiyon1(long x)
        {
            Console.WriteLine("CAGRILAN FONKSIYON : [ int fonksiyon1(long b); ]");
            return (int) x;
        }
    }
    class Sinif2
    {
        static void Main()
        {
            int int1 = 50;
            long lng1 = 50;
            Sinif1 nesne1 = new Sinif1();
            long sonuc1 = nesne1.fonksiyon1(int1);
            long sonuc2 = nesne1.fonksiyon1(lng1);
            long sonuc3 = nesne1.fonksiyon1(35);
            long sonuc4 = nesne1.fonksiyon1((long)35);



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

Çıktı:

```
CAGRILAN FONKSIYON : [ int fonksiyon1(int a); ]
CAGRILAN FONKSIYON : [ int fonksiyon1(long b); ]
CAGRILAN FONKSIYON : [ int fonksiyon1(int a); ]
CAGRILAN FONKSIYON : [ int fonksiyon1(long b); ]
```

- **II** ile belirtilen değişiklik yapıldığında **ISinif3** arayüzüne; **ISinif1**' den **fonksiyon1** adında, **int** tipinden tek bir argüman alan ve **int** tipinden değer döndüren bir üye ile **ISinif2**' den **fonksiyon1** adında, **int** tipinden tek bir argüman alan ve **long** tipinden değer döndüren bir üye olmak üzere toplamda iki farklı üye aktarılacaktır. **ISinif3** arayüzünden miras alan **Sinif1** sınıfı ise imzası verilen bu iki üyeyi de gerçeklemek durumunda kalacaktır. Ancak burada durum, **I** dekinden farklıdır. İsimleri ve argüman tipleri aynı ancak dönüş değer tipleri farklı olan birden çok sayıda fonksiyonu aynı sınıf içerisinde tanımlamak da mümkün olmayacaktır çünkü fonksiyon çağrısı yapıldığında hangi fonksiyonun görev almasının istendiğini açıkça belirtmek mümkün olmamaktadır. Böylesi bir değişiklik, gerçekleştirim işlemini çıkmaza sokacaktır.
- **III** ile belirtilen değişiklik yapıldığında **ISinif3** arayüzüne; **ISinif1**' den **fonksiyon1** adında, **int** tipinden tek bir argüman alan ve **int** tipinden değer döndüren bir üye ile **ISinif2**' den **fonksiyon2** adında, **int** tipinden tek bir argüman alan ve **int** tipinden değer döndüren bir üye olmak üzere toplamda iki farklı üye aktarılacaktır. **ISinif3** arayüzünden miras alan **Sinif1** sınıfı ise imzası verilen bu iki üyeyi de gerçeklemek durumunda kalacaktır.

Sınıflararası kalıtlamada olduğu gibi, arayüzler arasında yapılan kalıtlamada da döngüsel bir ilişki kurmak mümkün değilken mirasın kuşaktan kuşağa aktarılması mümkündür:

<pre> ... public interface ISinif1 : ISinif3 { int fonksiyon1(int a); } public interface ISinif2 : ISinif1 { int fonksiyon2(int b); } public interface ISinif3 : ISinif2 { int fonksiyon3(int c); } ... </pre> 	<pre> ... public interface ISinif1 { int fonksiyon1(int a); } public interface ISinif2 : ISinif1 { int fonksiyon2(int b); } public interface ISinif3 : ISinif2 { int fonksiyon3(int c); } ... </pre> 
--	--

Bir sınıfın, bir sınıf ve bir arayüzden miras alması da mümkündür:


```
using System;

namespace Uzay1
{
    public interface IArayuz1
    {
        int fonksiyon1(int a);
    }
    public class Sinif1
    {
        public int a = 84;

        public int fonksiyon2()
        {
            return a * a;
        }
    }
    public class Sinif2 : Sinif1, IArayuz1
    { // hem sınıftan hem arayuzden miras
        public int fonksiyon1(int i)
        {
            return i + i;
        }
    }
    public class Sinif3
    {
        static void Main()
        {
            Sinif2 nesne = new Sinif2();
            int sonuc1 = nesne.fonksiyon1(7);
            int sonuc2 = nesne.fonksiyon2();
            Console.WriteLine("1. Sonuc : {0}", sonuc1);
            Console.WriteLine("2. Sonuc : {0}", sonuc2);
            Console.ReadLine();
        }
    }
}
```

Çıktı:

```
1. Sonuc : 14
2. Sonuc : 7056
```