

7.1. NDP Temel Kavram II: Kalıtım (Inheritance)

Tıpkı doğada olduğu gibi (anne-babanın çocuklarına genetik özelliklerini devredebilmesi); kalıtım bir sınıfın izin verilen üyelerini başka sınıflara verebilme yeteneği olarak tanımlanabilir. (Bir sınıfın başka bir sınıfın üyelerini kendi üyesi gibi davranmalarını sağlayabilen özellik.) Burada ki temel amaç benzer kodları barındıran sınıfların ortak özellik/davranışlarını bir araya toplayarak kodun tekrar kullanılabilirliği (reuse) ve merkezi yönetimini sağlamaktır.

Burada kalıtım veren sınıf; taban (base), kalıtım alan sınıf ise türemiş sınıf (derived, çocuk sınıf) olarak adlandırılır. Taban sınıftan türemiş sınıfa doğru kalıtım verme ya da tam aksi olarak türemiş sınıftan taban sınıfa doğru ise miras alma gerçekleşir.

Türemiş sınıflar, taban sınıfların özelliklerini/davranışlarını alabildikleri gibi kendileri başka özellikler edinebilir, ya da almış oldukları özellik/davranışları geçersiz kılabilirler.

C++ birden çok sınıftan kalıtım almayı (multi inheritance) desteklerken, C# ve Java'da buna kısıtlama getirerek ancak tek bir sınıftan kalıtım almayı, ancak daha fazla gerek duyulan hallerde interface (arayüz) adı verilen yapılardan mümkün kılmıştır. Sınıflardan kalıtım C#'ta ":" operatörü ile sağlanırken Java'da extends anahtar kelimesi ile gerçekleştirilir. Teorik olarak kalıtım sonsuz seviyededir. Ancak tecrübeler 4 ila 6. kademedен sonra analizin ciddi şekilde tekrar incelenmesi gerektiğini söylemektedir. Temel Sınıf

```
class BaseClass
{
    public void EkranaYazdirBase()
    {
        Console.WriteLine("Temel Sınıf");
    }
}
```

Türemiş Sınıfa temel sınıftan kalıtım veriyoruz.

```
class DerivedClass : BaseClass
{
    public void EkranaYazdirDerived()
    {
        Console.WriteLine("Türemiş Sınıf");
    }
}
```

```

    }
}

```

```

class Program
{
    static void Main(string[] args)
    {
        BaseClass bc = new BaseClass();

        bc.EkranaYazdirBase();

        DerivedClass dc = new DerivedClass();
        dc.
    }
}

```

EkranaYazdirBase void BaseClass.EkranaYazdirBase()
 EkranaYazdirDerived
 Equals
 GetHashCode
 GetType
 ToString

DerivedClass'ta her ne kadar EkranaYazdirBase() isimli bir metod olmasa da BaseClass'tan kalıtım aldığından dolayı izin verilen tüm özellikler/davranışlar türeyen sınıfa geçmektedir.

Main kodları

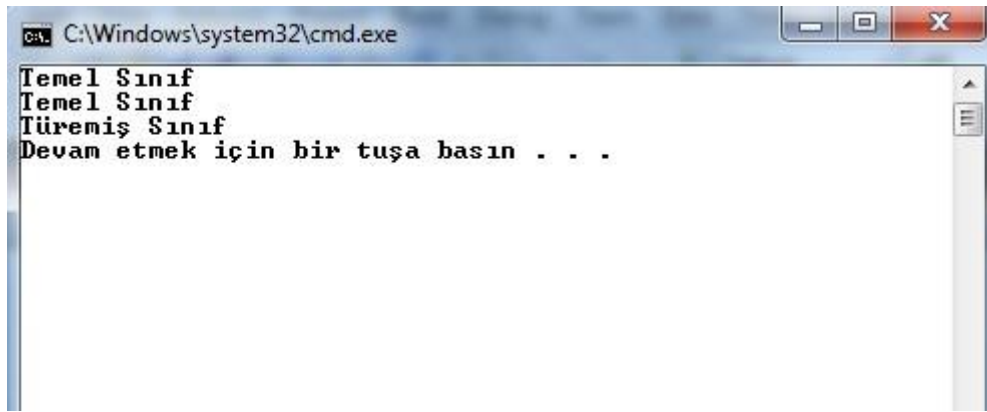
```

class Program
{
    static void Main(string[] args)
    {
        BaseClass bc = new BaseClass();

        bc.EkranaYazdirBase();

        DerivedClass dc = new DerivedClass(); dc.EkranaYazdirBase();
        dc.EkranaYazdirDerived();
    }
}

```



A screenshot of a Windows command prompt window. The title bar shows the path 'C:\Windows\system32\cmd.exe'. The window contains the following text: 'Temel Sınıf', 'Temel Sınıf', 'Türetilmiş Sınıf', and 'Devam etmek için bir tuşa basın . . .'. The text is displayed in a monospaced font.

7.2. Erişim Denetleyicilerin Kalıtım üzerindeki etkileri

```
class BaseClass
{
    public string adi;
    protected string soyAdi;
    private string meslek;
}
```

```
class DerivedClass : BaseClass
{
    public void EkranaYazdir()
    {
    }
}
```

public ile izin verilen alanlar alt sınıfta kullanılabilir.

```
class DerivedClass : BaseClass
```

```
{
```

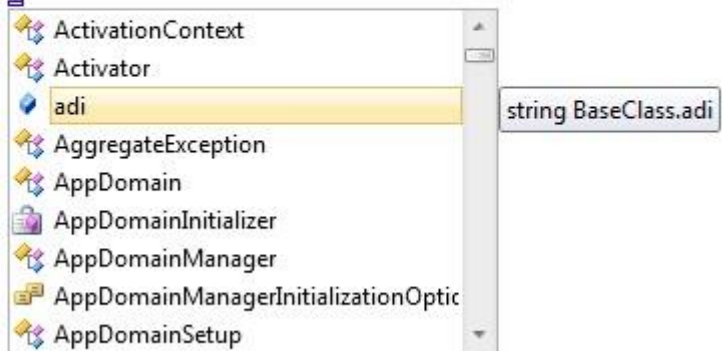
```
    public void EkranaYazdir()
```

```
    {
```

```
        Console.WriteLine("Adi" + a
```

```
    }
```

```
}
```



protected ile izin verilen alanlar alt sınıfta kullanılabilir.

```
class DerivedClass : BaseClass
```

```
{
```

```
    public void EkranaYazdir()
```

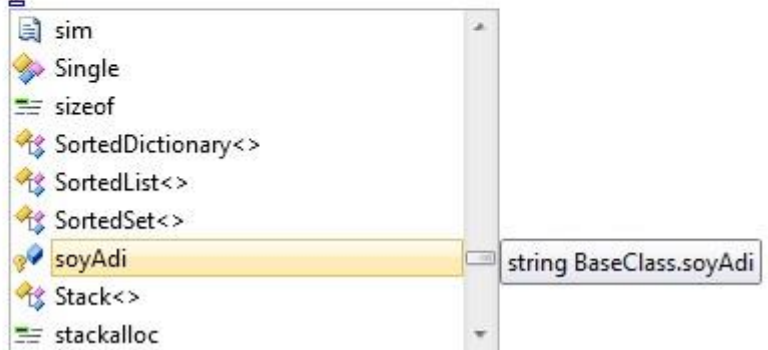
```
    {
```

```
        Console.WriteLine("Adi" + adi);
```

```
        Console.WriteLine("Soyadi" + s
```

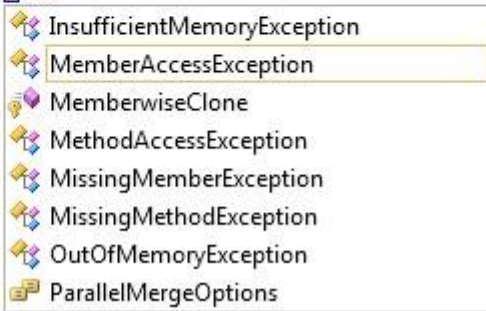
```
    }
```

```
}
```



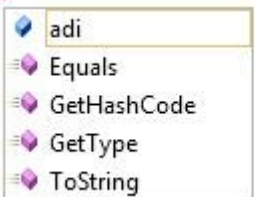
Temel sınıfta private olarak tanımlanan değerler alt sınıfta kullanılamamaktadır.

```
class DerivedClass : BaseClass
{
    public void EkranaYazdir()
    {
        Console.WriteLine("Adi" + adi);
        Console.WriteLine("Soyadi" + soyAdi);
        Console.WriteLine("Mesleği" + mes
    }
}
```



Temel sınıfta public olarak tanımlanan alanlar türemiş sınıf üzerinden diğer sınıflarda da kullanılabilir.

```
class Program
{
    static void Main(string[] args)
    {
        DerivedClass dr = new DerivedClass();
        dr.
    }
}
```



7.3. base anahtar kelimesi

Türemiş sınıftan temel sınıf üyelerine (alanlar, kurucular, metotlar vb.) base anahtar kelimesi ile erişilebilir.

BaseClass

```
class BaseClass
{
    public double a = 10;
    protected double b = 20;
    private double c = 30;
}
```

```
}
```

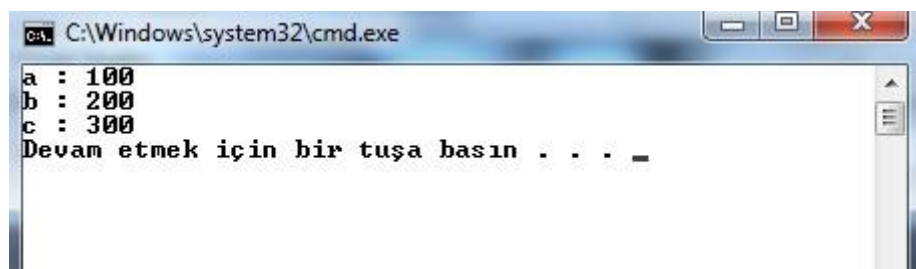
DerivedClass

```
class DerivedClass : BaseClass
{
    private double a = 100;
    public double b = 200;
    protected double c = 300;
    public void EkranaYazdir()
    {
        Console.WriteLine("a : " + a);
        Console.WriteLine("b : " + b);
        Console.WriteLine("c : " + c);
    }
}
```

Main kodları;

```
class Program
{
    static void Main(string[] args)
    {
        DerivedClass dr = new DerivedClass();
        dr.EkranaYazdir();
    }
}
```

Ekran çıktısı;



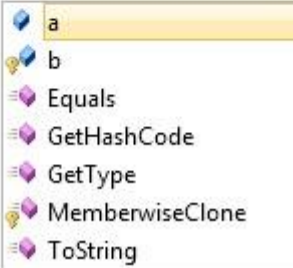
The screenshot shows a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The output of the program is displayed as follows:

```
a : 100
b : 200
c : 300
Devam etmek için bir tuşa basın . . . _
```

Hem temel sınıfta hem de türemiş sınıfta aynı isimdeki değişkenler var ise erişim denetleyicileri değişse bile en son değer olarak türemiş sınıftaki değerlerini almaktadır. Bu tür durumlarda bazen ana sınıftaki değerleri ulaşmak istenilirse **base** anahtar kelimesi kullanılarak temel sınıfın izin verilen üyelerine ulaşılabilir.

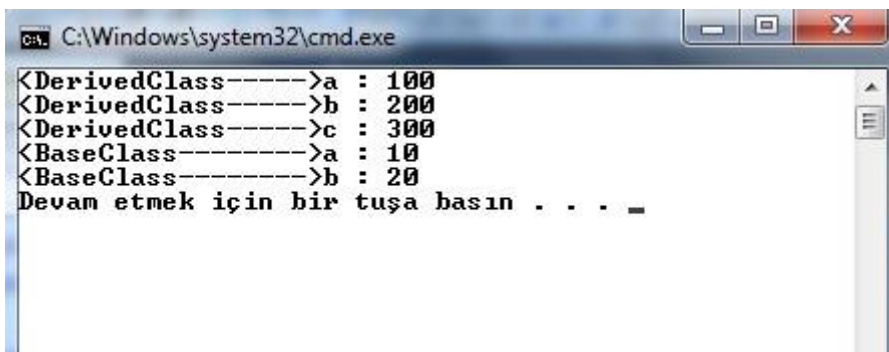
```
class DerivedClass : BaseClass
{
    private double a = 100;
    public double b = 200;
    protected double c = 300;

    public void EkranaYazdir()
    {
        Console.WriteLine("a : " + a);
        Console.WriteLine("b : " + b);
        Console.WriteLine("c : " + c);
        Console.WriteLine("a : " + base.
    }
}
```



DerivedClass kodlarımız;

Ekran çıktısı;



```
C:\Windows\system32\cmd.exe
<DerivedClass----->a : 100
<DerivedClass----->b : 200
<DerivedClass----->c : 300
<BaseClass----->a : 10
<BaseClass----->b : 20
Devam etmek için bir tuşa basın . . .
```

```

class DerivedClass : BaseClass
{
    private double a = 100; public double b = 200;
    protected double c = 300;

    public void EkranaYazdir()
    {
        Console.WriteLine("<DerivedClass----->a : " + a);
        Console.WriteLine("<DerivedClass----->b : " + b);
        Console.WriteLine("<DerivedClass----->c : " + c);
        Console.WriteLine("<BaseClass----->a : " + base.a);
        Console.WriteLine("<BaseClass----->b : " + base.b);
    }
}

```

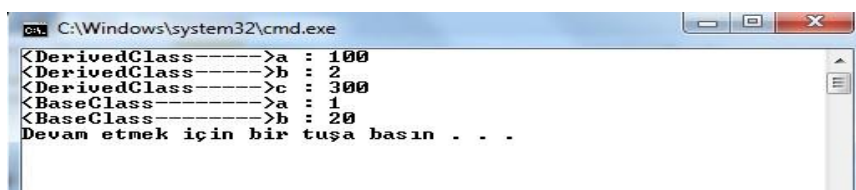
Main kodlarımızı aşağıdaki şekilde değiştirelim.

```

class Program
{
    static void Main(string[] args)
    {
        BaseClass br = new BaseClass();
        br.a = 1000;
        DerivedClass dr = new DerivedClass();
        dr.a = 1;
        dr.b = 2;
        dr.EkranaYazdir();
    }
}

```

Ekran çıktımız;



```

C:\Windows\system32\cmd.exe
<DerivedClass----->a : 100
<DerivedClass----->b : 2
<DerivedClass----->c : 300
<BaseClass----->a : 1
<BaseClass----->b : 20
Devam etmek için bir tuşa basın . . .

```

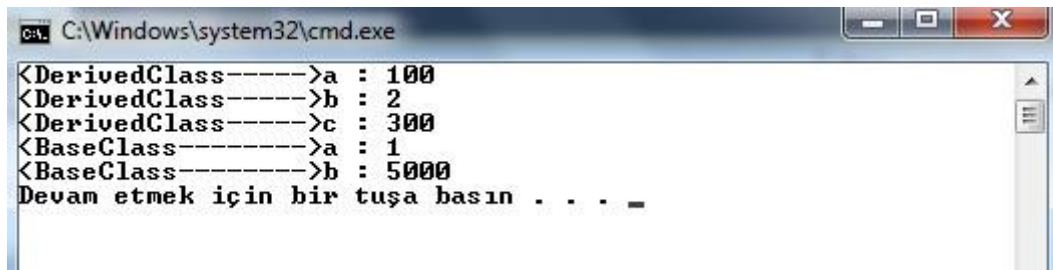

Bazı deęerler deęiřirken bazıları deęiřmemiřtir. Bunun nedenlerini incelersek;

Sınıf Adı	Alan	Eski Deęer	Yeni Deęer	Sebep
DerivedClass	a	100	100	DerivedClass'ta private olarak tanımlandığından deęiřtirilmeye alıřılan BaseClass'ın a deęeri.
DerivedClass	b	200	2	DerivedClass'ta public olarak tanımlandığından deęiřme var.
DerivedClass	c	300	-	DerivedClass'ta protected olarak tanımlandığından main iersinden eriřilemez.
BaseClass	a	1000	1	İlk etapta 1000 deęeri atansa da BaseClass'taki özellikler DerivedClass'a gemektedir. DerivedClass'tan üretilen yeni nesne ile tekrar deęer verilmektedir.
BaseClass	b	20	20	BaseClass'ta protected olarak tanımlı olduğundan deęeri sadece DerivedClass'tan deęiřtirilebilir.
BaseClass	c	30	-	BaseClass'ta private olarak tanımlı

DerivedClass kodlarımıza base.b=5000 ifadesini ekleyelim;

```
class DerivedClass : BaseClass
{
    private double a = 100;
    public double b = 200;
    protected double c = 300;
    public void EkranaYazdir()
    {
        base.b = 5000;
        Console.WriteLine("<DerivedClass----->a : " + a);
        Console.WriteLine("<DerivedClass----->b : " + b);
        Console.WriteLine("<DerivedClass----->c : " + c);
        Console.WriteLine("<BaseClass----->a : " + base.a);
        Console.WriteLine("<BaseClass----->b : " + base.b);
    }
}
```

BaseClass b değeri 5000 olarak yazdırılacaktır.



```
C:\Windows\system32\cmd.exe
<DerivedClass----->a : 100
<DerivedClass----->b : 2
<DerivedClass----->c : 300
<BaseClass----->a : 1
<BaseClass----->b : 5000
Devam etmek için bir tuşa basın . . . -
```

7.4. Metotlarda kalıtım

BaseClass

```
class BaseClass
{
    public double a = 10;
    protected double b = 20;
    private double c = 30;

    public void EkranaYazdirBaseClass()
    {
        Console.WriteLine("<BaseClass----->a : " + a);
        Console.WriteLine("<BaseClass----->b : " + b);
    }
}
```

```

        Console.WriteLine("<BaseClass----->c : " + c);
    }
}

```

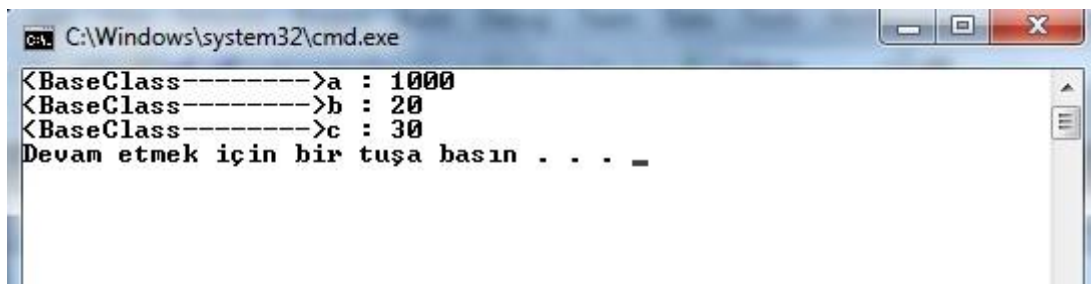
Main kodları

```

class Program
{
    static void Main(string[] args)
    {
        BaseClass br = new BaseClass();
        br.a = 1000;
        br.EkranaYazdirBaseClass();
    }
}

```

a değişkenine en son 1000 değeri atandığından sonuç ekranında değeri 1000 olarak görülmektedir.



DerivedClass kodlarımız;

```

class DerivedClass : BaseClass
{
    private double a = 100;
    public double b = 200;
    protected double c = 300;

    public void EkranaYazdirDerivedClass()
    {
        base.b = 5000;
        Console.WriteLine("<DerivedClass----->a : " + a);
        Console.WriteLine("<DerivedClass----->b : " + b);
    }
}

```

```

        Console.WriteLine("<DerivedClass----->c : " + c);
        Console.WriteLine("<BaseClass----->a : " + base.a);
        Console.WriteLine("<BaseClass----->b : " + base.b);
    }
}

```

Main kodlarının yeni hali;

```

class Program
{
    static void Main(string[] args)
    {
        BaseClass br = new BaseClass();
        br.a = 1000;
        br.EkranaYazdirBaseClass();

        Console.WriteLine();
        Console.WriteLine("*****");
        Console.WriteLine();

        DerivedClass dr = new DerivedClass();
        dr.a = -1; dr.b = -2;
        dr.EkranaYazdirBaseClass(); // kalıtım yoluyla geçmiştir
        dr.EkranaYazdirDerivedClass();
    }
}

```

Ekran görünümümüz ise aşağıdaki şekildedir.

```

C:\Windows\system32\cmd.exe
<BaseClass----->a : 1000
<BaseClass----->b : 20
<BaseClass----->c : 30
*****
<BaseClass----->a : -1
<BaseClass----->b : 20
<BaseClass----->c : 30
<DerivedClass----->a : 100
<DerivedClass----->b : -2
<DerivedClass----->c : 300
<BaseClass----->a : -1
<BaseClass----->b : 5000
Devam etmek için bir tuşa basın . . . _

```

Temel sınıfta ve türeyen sınıfta aynı isimde metotların bulunması durumunda ise türemiş sınıf temel sınıfın metotlarını geçersiz kılarak kendi tanımlamalarını kullanır. Ancak derleyici size yine de uyarı verecektir.

TemelSinif;

```
class TemelSinif
{
    public void Hesapla()
    {
        Console.WriteLine("TemelSinif");
    }
}
```

TuremisSinif

```
class TuremisSinif : TemelSinif
{
    public void Hesapla()
    {
        Console.WriteLine("TuremisSinif");
    }
}
```

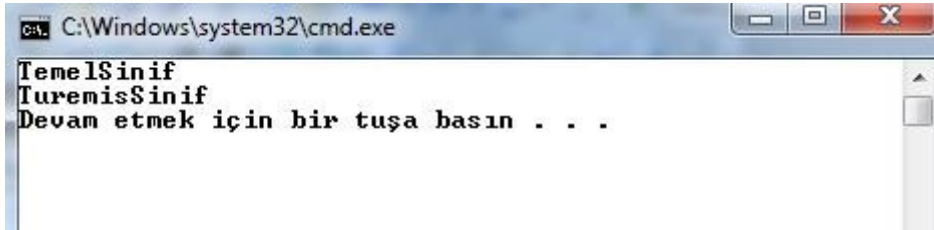
Main Kodları

```
class Program
{
    static void Main(string[] args)
    {
        TemelSinif ts = new TemelSinif();
        ts.Hesapla();

        TuremisSinif tus = new TuremisSinif();
        tus.Hesapla();
    }
}
```

```
}  
}
```

Ekran çıktısı;



```
C:\Windows\system32\cmd.exe  
TemelSinif  
TuremisSinif  
Devam etmek için bir tuşa basın . . .
```

Derleyicin vereceği bu uyarıyı türeyen sınıftaki metodun başına new anahtar sözcüğü getirerek aşabiliriz.

```
class TuremisSinif : TemelSinif  
{  
    new public void Hesapla()  
    {  
        Console.WriteLine("TuremisSinif");  
    }  
}
```

Metotların abstract ya da virtual olması farklı bir durumdur.

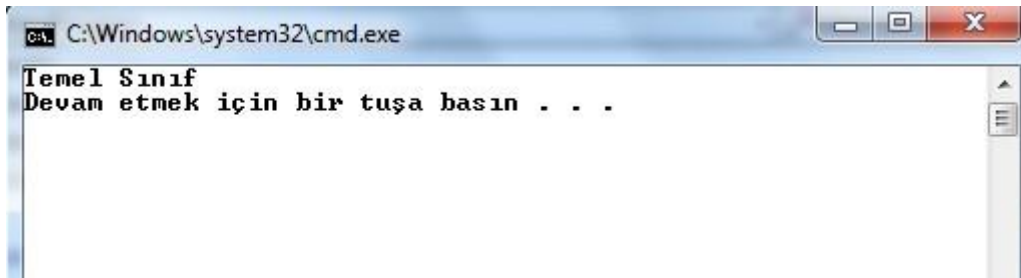
7.5. Kurucularda Kalıtım

Durum 1:

```
class A
{
    public A()
    {
        Console.WriteLine("Temel Sınıf");
    }
}

class B : A
{
}

class Program
{
    static void Main(string[] args)
    {
        B b = new B();
    }
}
```



Sonuç: Türemiş sınıflardan bir nesne oluşturulurken arka planda kalıtım veren sınıftan (base class) otomatik olarak bir nesne oluşturulur ve kalıtım bu nesne üzerinden gerçekleştirilir. Burada dikkat edilmesi gereken konu türemiş sınıfın kurucusunun çalışmasından önce temel sınıfın kurucusunun çalışması gerektirir.

Durum 2: Türemiş sınıf temel sınıf kurucu yapılandırması.

```
class A
{
    public A(int a, int b)
    {
        Console.WriteLine(a + b);
    }
}

class B : A
{
    public B(int a)
    {
        Console.WriteLine(a);
    }
}

class Program
{
    static void Main(string[] args)
    {
        B b = new B(5);
    }
}
```

Error List		
1 Error 0 Warnings 0 Messages		
	Description	File
1	'Kurucular.A' does not contain a constructor that takes 0 arguments	B.cs

B sınıfı kurulurken A sınıfından kalıtım aldığından A sınıfının kurucusu çağrılmaktadır. Ancak A sınıfının kurucusunun istediği şartlar sağlanamamaktadır. A sınıfından otomatik olarak oluşacak nesnenin kurucusunun istediği başlangıç değerleri B sınıfından çağrılacak kurucudan sağlanmak zorundadır. Bu durumda B sınıfının kurucuları A sınıfının kurucularına base anahtar kelimesi kullanarak gerekli şartları sağlayabilirler.

Yukarıdaki örneği aşağıdaki biçimde değiştirelim.

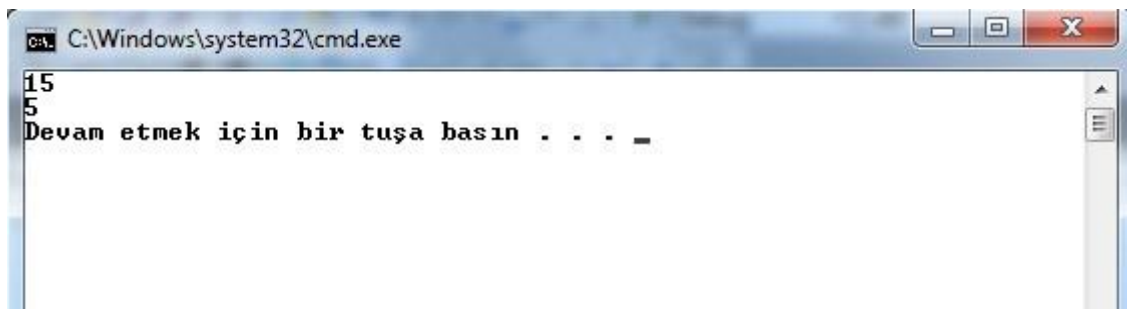
```
class A
{
    public A(int a, int b)
    {
```



```

        Console.WriteLine(a + b);
    }
}
class B : A
{
    public B(int a) : base(a, 10)
    {
        Console.WriteLine(a);
    }
}
class Program
{
    static void Main(string[] args)
    {
        B b = new B(5);
    }
}

```



Kalıtımda başka bir konu ise kimin kimden türeyeceğidir. Uygulamalarda genel kabul; kurucudaki parametre sayısının fazla olanının temel sınıf olması yönündedir.

7.6. Kalıtımın engellenmesi

NDP de her sınıf diğer sınıf ya da ara yüzlerden¹ kalıtım alınabilmekteydi. Bazı durumda bunun engellenmesi gerekebilir. C#’ta **sealed** anahtar kelimesi ile bir sınıfın kalıtım vermesi engellenebilir. (Static² sınıflar doğası gereği kalıtım almazlar ve vermezler.)

¹ Arayüz (Interface) kavramı soyut sınıflar konusunda işlenecektir.

² Static kavramı static classlar konusunda işlenecektir.

```
public class Calisan
{
    public string adi = null;
    public string soyadi = null;
}

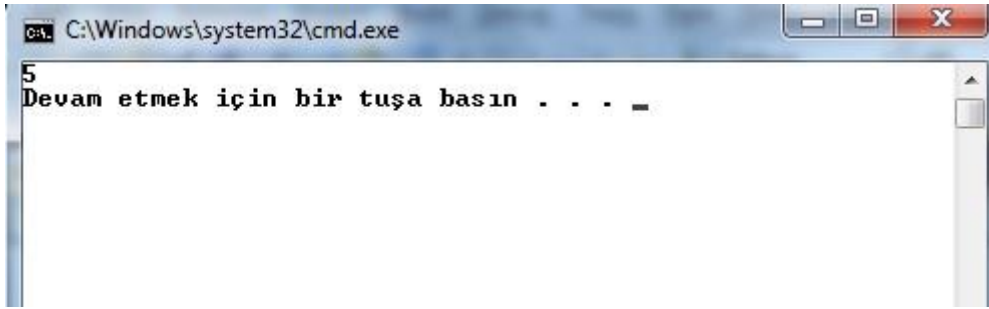
public class Mudur : Calisan
{
    public string departman = null;
}

sealed public class GenelMudur : Mudur
{
    public double MaasHesapla()
    {
        return 5;
    }
}

class Program
{
    static void Main(string[] args)
    {
        Mudur b = new Mudur();

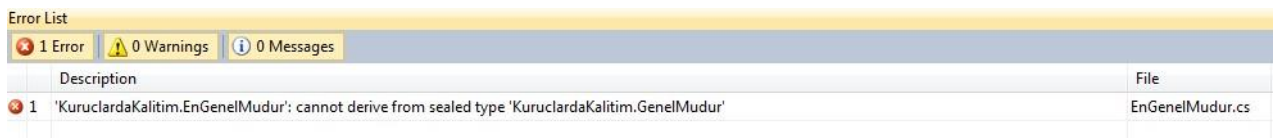
        b.adi = "Ümit";
        b.soyadi = "Kocabıçak";
        b.departman = "Bilgisayar";

        GenelMudur gm = new GenelMudur();
        gm.adi = ".....";
        gm.soyadi = ".....";
        gm.departman = "*****";
        Console.WriteLine(gm.MaasHesapla());
    }
}
```



Programımıza yeni bir sınıf ekleyim GenelMudur sınıfından kalıtım aldırmaı deneyelim.

```
public class EnGenelMudur:GenelMudur
{
}
```



sealed sınıflardan kalıtım alınamaz şeklinde bir hata alırsınız.

7.7. virtual anahtar kelimesi

Temel sınıflarda virtual ile işaretlenmiş yapılar türemiş sınıflara 2 seçenek sunar.

- Eğer türemiş sınıf bu metodu yazmaz ise kalıtım aldığı sınıfın metodunu kullanır.
- Türemiş sınıfı bu metodu kendine özgü şekilde yazmak isterse **override** anahtar kelimesi ile ana sınıfın metodunu geçersiz kılıp(üstüne yazma) kendi yazdığı işlemleri gerçekleştirir.

```
public class GenelSinif
{
    public double a;
    public double b;

    public virtual void EkranaYazdir()
    {
        Console.WriteLine(a + b);
    }
}
```

```

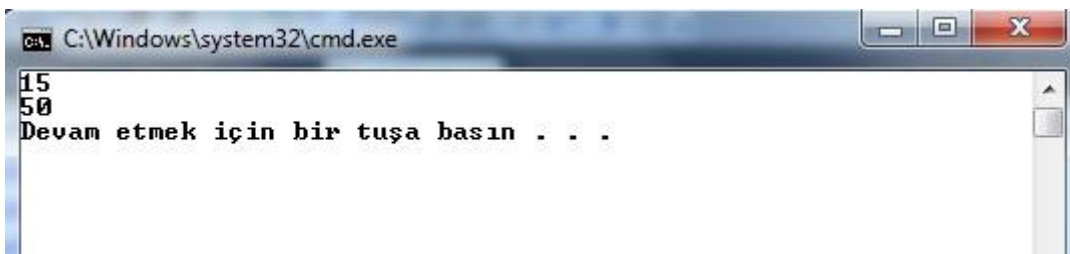
    }

    public virtual void EkrandaGoster()
    {
        Console.WriteLine(a * b);
    }
}

class TuremisSinif : GenelSinif
{
}

class Program
{
    static void Main(string[] args)
    {
        TuremisSinif ts = new TuremisSinif();
        ts.a = 5; ts.b = 10;
        ts.EkranaYazdir();
        ts.EkrandaGoster();
    }
}

```

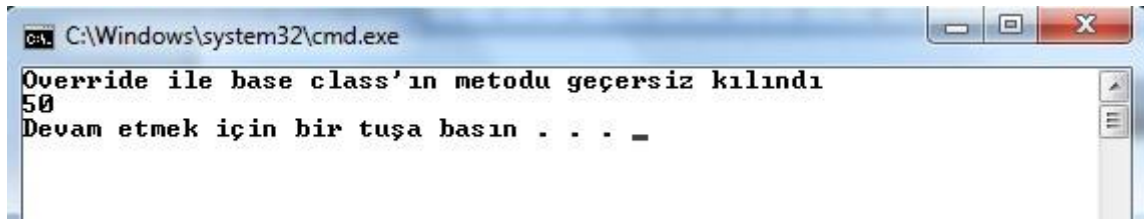


TuremisSinif kodlarımızı aşağıdaki şekilde değiştirelim.

```

class TuremisSinif : GenelSinif
{
    public override void EkranaYazdir()
    {
        Console.WriteLine("Override ile base class'ın metodu geçersiz
kılındı");
    }
}

```



Örnek Uygulama: Nokta, daire, küre ve silindirin alanlarını hesaplayan program.

(Koordinatlar dikkate alınmayacaktır.)

```
class Nokta
{
    public double AlanHesapla()
    {
        return 0;
    }
}

class Daire
{
    private double r;
    public Daire(double r)
    {
        this.r = r;
    }
    public double AlanHesapla()
    {
        return Math.PI * r * r;
    }
}

class Kure
{
    private double r;

    public Kure(double r)
    {
        this.r = r;
    }

    public double AlanHesapla()
    {
        return 4 * Math.PI * r * r;
    }
}
```

```

    }

class Silindir
{
    private double r;
    private double h;

    public Silindir(double r, double h)
    {
        this.r = r;
        this.h = h;
    }

    public double AlanHesapla()
    {
        return 2 * Math.PI * r * r + 2 * Math.PI * r * h;
    }
}

class Program
{
    static void Main(string[] args)
    {
        // Yarıçap ve Yükseklik Değerlerinin okunması
        Console.Write("Yarıçap : ");
        double yariCap = Convert.ToDouble(Console.ReadLine());
        Console.Write("Yukseklik: ");
        double yukseklik = Convert.ToDouble(Console.ReadLine());

        // Yeni nesnelerin oluşturulması
        Nokta yeniNokta = new Nokta();
        Daire yeniDaire = new Daire(yariCap);
        Kure yeniKure = new Kure(yariCap);
        Silindir yeniSilindir = new Silindir(yariCap, yukseklik);

        // Ekranda Gösterilmesi
        Console.WriteLine("Noktanın Alanı    = {0:F2}",
yeniNokta.AlanHesapla());
        Console.WriteLine("Dairenin Alanı    = {0:F2}",
yeniDaire.AlanHesapla()); Console.WriteLine("Kürenin Alanı    = {0:F2}",
yeniKure.AlanHesapla());
        Console.WriteLine("Silindirin Alanı = {0:F2}",
yeniSilindir.AlanHesapla());
    }
}

```

Adım 1: Kodun yeniden kullanılabilirliğinin artırılması ve kodda merkezi yönetim sağlanması.

Silindir 2, Daire ve Küre tek, Nokta sınıfı ise parametresiz kurucu istemektedir. Yeni bir sınıf oluşturarak parametresiz ve 2 parametrelili kurucusunu yazalım.

```
class SekilBase
{
    protected double r;
    protected double h;

    public SekilBase()
    {
    }

    public SekilBase(double r, double h)
    {
        this.r = r;
        this.h = h;
    }

    public virtual double AlanHesapla()
    {
        return 0;
    }
}

class Nokta : SekilBase
{
}

class Daire : SekilBase
{
    public Daire(double r) : base(r, 0)
    {
    }

    public override double AlanHesapla()
    {
        return Math.PI * r * r;
    }
}

class Kure : SekilBase
{
}
```

```

        public Kure(double r) : base(r, 0)
        {
        }

        public override double AlanHesapla()
        {
            return 4 * Math.PI * r * r;
        }
    }

    class Silindir : SekilBase
    {
        public Silindir(double r, double h) : base(r, h)
        {
        }
        public override double AlanHesapla()
        {
            return 2 * Math.PI * r * r + 2 * Math.PI * r * h;
        }
    }

```

Main tarafındaki kodlarımızı hiç değiştirmeden programı çalıştırdığımızda;



```

C:\Windows\system32\cmd.exe
Yarıçap : 5
Yükselik: 6
Noktanın Alanı = 0,00
Dairenin Alanı = 78,54
Kürenin Alanı = 314,16
Silindirin Alanı = 345,58
Devam etmek için bir tuşa basın . . . _

```

sonucunu elde ederiz.

Adım 2: Main tarafında sürekli olarak alanları ekrana yazdırma kodları yer almaktadır. SekilBase sınıfına ekrana yazdırma kodlarını çekelim. Bu durumda dışardan public olarak belirtilen AlanHesapla() metodunun da erişim seviyesini tüm sınıflarda protected düzeye indirerek sarmalamayı da gerçekleştirmiş oluruz. Yeni kodlarımız;

```
class SekilBase
{
    protected double r;
    protected double h;
    public SekilBase()
    { }
    public SekilBase(double r, double h)

    {
        this.r = r;
        this.h = h;
    }
    protected virtual double AlanHesapla()
    {
        return 0;
    }
    public void EkranaYazdir()
    {
        Console.WriteLine(this.GetType().Name + " Alanı = {0:F2}",
this.AlanHesapla());
    }
}

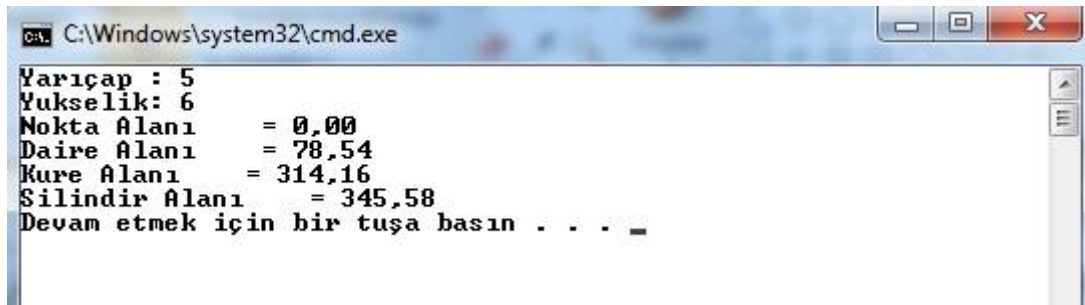
class Nokta : SekilBase
{
}
class Daire : SekilBase
{
    public Daire(double r) : base(r, 0)
    {
    }
    protected override double AlanHesapla()
    {
        return Math.PI * r * r;
    }
}
class Kure : SekilBase
{
    public Kure(double r) : base(r, 0)
    {
    }
}
```

```

        protected override double AlanHesapla()
        {
            return 4 * Math.PI * r * r;
        }
    }
    class Silindir : SekilBase
    {
        public Silindir(double r, double h) : base(r, h)
        {
        }
        protected override double AlanHesapla()
        {
            return 2 * Math.PI * r * r + 2 * Math.PI * r * h;
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            // Yarıçap ve Yükseklik Değerlerinin okunması
            Console.Write("Yarıçap : ");
            double yariCap = Convert.ToDouble(Console.ReadLine());
            Console.Write("Yukseklik: ");
            double yukseklik = Convert.ToDouble(Console.ReadLine());

            // Yeni nesnelerin oluşturulması
            Nokta yeniNokta = new Nokta();
            Daire yeniDaire = new Daire(yariCap);
            Kure yeniKure = new Kure(yariCap);
            Silindir yeniSilindir = new Silindir(yariCap, yukseklik);
            // Ekranda Gösterilmesi
            yeniNokta.EkranaYazdir();
            yeniDaire.EkranaYazdir();
            yeniKure.EkranaYazdir();
            yeniSilindir.EkranaYazdir();
        }
    }
}

```



```
C:\Windows\system32\cmd.exe
Yarıçap : 5
Yukseklik: 6
Nokta Alanı = 0,00
Daire Alanı = 78,54
Küre Alanı = 314,16
Silindir Alanı = 345,58
Devam etmek için bir tuşa basın . . . _
```

Adım 3: Main tarafındaki kodlarımız mainin gerçek görevinden uzaklaştırmaktadır. Temel hedeflerimizden biri olan çalışan kodların değişmeden programımızın gelişebilmesi sağlanamamaktadır.

Main içerisinde yer alan kodlarımızı başka bir sınıfa(GenelSınıf) çekelim;

```
class GenelSınıf
{
    private double yariCap;
    private double yukseklik;

    public void EkranOku()
    {
        Console.Write("Yarıçap : ");
        yariCap = Convert.ToDouble(Console.ReadLine());
        Console.Write("Yukseklik: ");
        yukseklik = Convert.ToDouble(Console.ReadLine());
    }
    public void NesneOlustur()
    {
        Nokta yeniNokta = new Nokta();
        Daire yeniDaire = new Daire(yariCap);
        Kure yeniKure = new Kure(yariCap);
        Silindir yeniSilindir = new Silindir(yariCap, yukseklik);

        // Ekranda Gösterilmesi
        yeniNokta.EkranaYazdir();
        yeniDaire.EkranaYazdir();
        yeniKure.EkranaYazdir();
        yeniSilindir.EkranaYazdir();
    }
}
class Program
{
    static void Main(string[] args)
```

```

    {
        GenelSinif gs = new GenelSinif();
        gs.EkranOku();
        gs.NesneOlustur();
    }
}

```

Main içerisinde GenelSinif'ın üyeleriyle muhatap olmamak için GenelSinif içerisindeki metotları kurucunun içerisine çekebiliriz.

```

class GenelSinif
{
    private double yariCap;
    private double yukseklik;
    public GenelSinif()
    {
        this.EkranOku();
        this.NesneOlustur();
    }
    private void EkranOku()
    {
        Console.Write("Yarıçap : ");
        yariCap = Convert.ToDouble(Console.ReadLine());
        Console.Write("Yukseklik: ");
        yukseklik = Convert.ToDouble(Console.ReadLine());
    }

    private void NesneOlustur()
    {
        Nokta yeniNokta = new Nokta();
        Daire yeniDaire = new Daire(yariCap);
        Kure yeniKure = new Kure(yariCap);
        Silindir yeniSilindir = new Silindir(yariCap, yukseklik);

        // Ekranda Gösterilmesi
        yeniNokta.EkranaYazdir();
        yeniDaire.EkranaYazdir();
        yeniKure.EkranaYazdir();
        yeniSilindir.EkranaYazdir();
    }
}

class Program
{

```

```

static void Main(string[] args)
{
    GenelSinif gs = new GenelSinif();
}
}

```

Adım 4: Yeni bir şeklin(dikdörtgenin) alanın hesaplanmasını sistemimize eklemek istersek;

```

class Dikdortgen : SekilBase
{
    public Dikdortgen(double en, double boy)
        : base(en, boy)
    {
    }
    protected override double AlanHesapla()
    {
        return r * h;
    }
}

class GenelSinif
{
    private double yariCap;
    private double yukseklik;

    public GenelSinif()
    {
        this.EkranOku();
        this.NesneOlustur();
    }

    private void EkranOku()
    {
        Console.Write("Yarıçap : ");
        yariCap = Convert.ToDouble(Console.ReadLine());
        Console.Write("Yukseklik: ");
        yukseklik = Convert.ToDouble(Console.ReadLine());
    }

    private void NesneOlustur()
    {
        Nokta yeniNokta = new Nokta();
    }
}

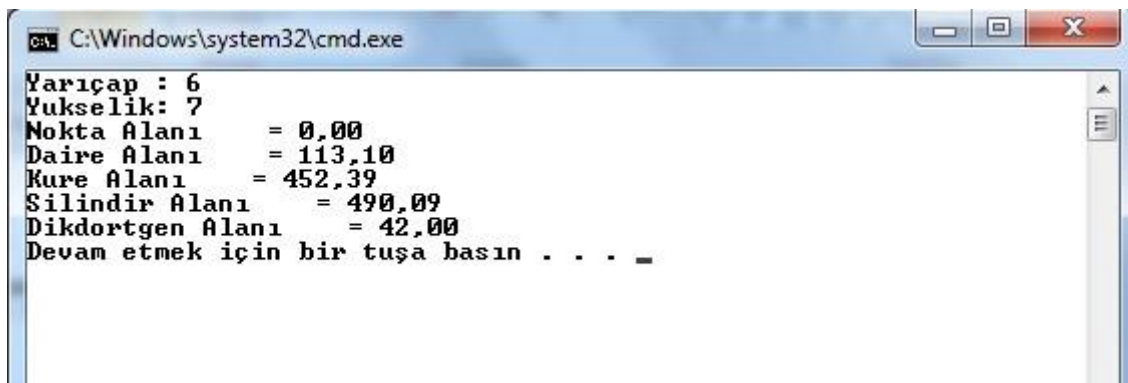
```

```

        Daire yeniDaire = new Daire(yariCap);
        Kure yeniKure = new Kure(yariCap);
        Silindir yeniSilindir = new Silindir(yariCap, yukseklik);
        Dikdortgen yeniDikdortgen = new Dikdortgen(yariCap,
yukseklik);

        // Ekranda Gösterilmesi
        yeniNokta.EkranaYazdir();
        yeniDaire.EkranaYazdir();
        yeniKure.EkranaYazdir();
        yeniSilindir.EkranaYazdir();
        yeniDikdortgen.EkranaYazdir();
    }
}

```



```

C:\Windows\system32\cmd.exe
Yarıçap : 6
Yukseklik: 7
Nokta Alanı      = 0,00
Daire Alanı      = 113,10
Kure Alanı       = 452,39
Silindir Alanı   = 490,09
Dikdortgen Alanı = 42,00
Devam etmek için bir tuşa basın . . . -

```

Dikkat edilirse Main kodlarımıza ve diğer sınıflarımıza hiçbir değişikliğe gitmeden sistemimize yeni şekillerin alanlarını hesaplayan sınıflar ekleyebildik.