8.1. Arayüzler (Interface)

C++ var olan birden çok sınıftan kalıtım alınabilme (çoklu kalıtım, multi inheritance) C# ve Java'da terkedilmiş ancak çoklu kalıtımı sağlayacak başka bir yapı Interface yapısı geliştirilmiştir.

Sınıflar için bir rehber olan arayüzler, birbirleri arasında kalıtım olmayan sınıflara bazı mecburiyetler bırakır. Sınıflar arası bir kontrat olarak da düşünebiliriz. Yazılımın genişlemesi birçok karmaşaya yol açabilir. Onlarca yüzlerce sınıf içerisinde hangisi ne işi yapar, nereden hangi yapıları almıştır, hangi görevleri yerine getirecektir şeklindeki sorularla muhatap olmaktadır.

Bunlarla beraber yeni oluşturulacak bir sınıfın içereceği yapının nasıl şekillenmesi gerektiğini de arayüzler aracılığıyla bildirebiliriz. Arayüzler interface anahtar kelimesi ile bildirilir. Genel kabul olarak da arayüz olduğunun isminden anlaşılabilmesi için isminden önce I(büyük harf ı) kullanılır.

```
interface IOrnek
    {

    }
```

Arayüzler diğer sınıf ya da arayüzlere kalıtım vermek için kullanılır. Yeni bir örneği (nesne) oluşturulamaz. Bunun en büyük nedeni var olmalarının gereği diğer interface ya da sınıflara rehber olmaları veya kontrat sağlamaları içindir. Zaten gövdelerinde kurucu bulundurmazlar.
Aşağıda ki kod bloğu derleme zamanı hatası üretecektir.

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

8.1.1. Arayüzlerin üyeleri

Arayüzler bünyelerinde;

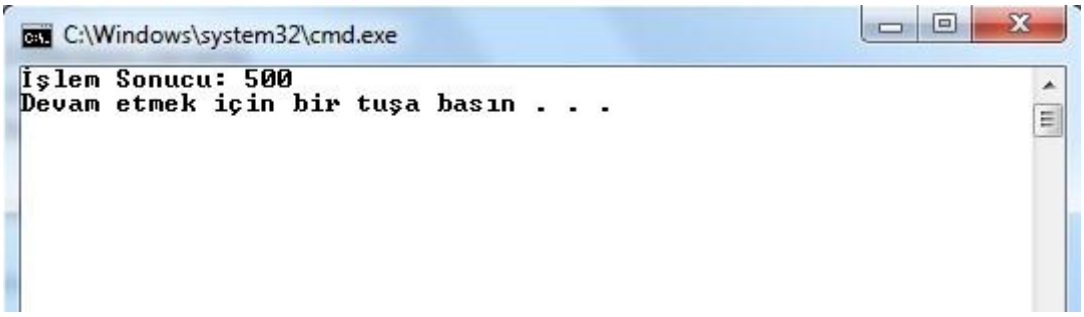
- Metotlar
- Özellikler ve indexer (propertylerin özel hali, sadece C#'ta geçerli)
- Olaylar

bulundururlar. Ancak metotların ya da propertylerin gövdeleri boştur. Sınıf üyelerinden alanların bulunması kendi görevleri açısından uygun değildir. Ayrıca kurucu ve yıkıcılarda bulunamazlar.

```
interface IOrnek
    {
        double X { get; set; }
        double Y{get;set;}

        double Hesapla();
    }
```

```
class Deneme : IOrnek
    {

        private double _x; private double _y;

        public double X
        {
            get
            {
                return _x;
            }
            set
            {
                _x = value;
            }
        }

        public double Y
        {
            get
            {
                return _y;
            }
            set
```

```
            {
                _y = value;
            }
        }
        public double Hesapla()
        {
            return _x * _y;
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            Deneme d = new Deneme();
            d.X = 100;
            d.Y = 5;

            Console.WriteLine("İşlem Sonucu: " + d.Hesapla());
        }
    }
```



Örnek: Bir arayüzde bulunabilecek tüm yapılar;

```
// property
        string Adı
        {
            get; set;
        }

        // Yalnız okunabilir özellik tanımı.
        string Soyadı
        {
            get;
        }

        // Yalnızca yazılabilir özellik tanımı
        int Yas
        {
            set;
        }

        //Geri dönüşü integer olan ve 2 integer parametre isteyen metot
tanımı
        int topla(int a, int b);

        // Geri dönüş değeri ya da parametre almayan metot tanımı
        void yaz();

        // Bir indeksleyici tanımı. Sadece C#'ta geçerlidir.
        string this[int index]
        {
            get; set;
        }
    }
```

Örneklerden de anlaşılacağı üzere arayüzler birer rehber ya da kontrat olduğundan yazılacak özellik, metot vb. Üyelerin içleri boş bırakılır. Amaç sınıflara bir takım işleri yapmayı hatırlatmaktır.

8.1.2. Arayüzlerin özellikleri

- Bir arayüzün tüm üyeleri public kabul edilir.
- Diğer yandan bir arayüz üyesi public olarak da tanımlayamayız. Çünkü zaten varsayılan olarak bütün üyeler public tanımlanmış kabul edilir.
- Bir arayüz, bir yapı(struct)'dan veya bir sınıf(class)'tan kalıtımla türetilemez. Ancak, bir arayüzü başka bir arayüzden veya arayüzlerden kalıtımsal olarak türetebiliriz.
- Arayüz elemanlarını static olarak tanımlayamayız.
- Arayüzlerin uygulandığı sınıflar, arayüzde tanımlanan bütün üyeleri kullanmak zorundadır.[1]

7.2.3. Arayüzler ve Çoklu kalıtım

C# ta sınıflar ancak bir sınıftan kalıtım alabilirken, dileği sayıda arayüzden kalıtım alabilirler. Burada önemli olan kalıtım veren sınıf ya da arayüzlerin üyelerinin isimlerinin çakışmamasıdır. Çakışması durumunda sınıfın bunu bir defa yazması yeterli olacaktır. Benzeri isimdeki üyelerin bulunması halinde ise ya başlarına arayüzün adı belirtilerek yazılır ya da nesne oluşturulurken hangi arayüzün ki kullanılacağı tip dönüşümü yapılarak kullanılır.

Örnek: Sipariş Modülü

```
interface IMusteri
    {
        string Ad { get; }
        string Soyad { get; set; }
        string Id { get; }
        void EkranaYazdir();
    }
interface ISiparis
    {
        int SiparisID { get; set; }
        string UrunAdi { get; set; }
        double Fiyat { get; set; }
        int Miktar { get; set; }
        void EkranaYazdir();
    }
```

[1] http://www.csharpnedir.com/articles/read/?filter=&author=&cat=cs&id=185&title=Aray%C3%BCz%28Interface%29 %20Kullan%C4%B1m%C4%B1na%20Giri%C5%9F

şeklinde arayüzlerimizi belirtebiliriz. Bu sistem daha ayrıntılı düşünüldüğünden IMusteri arayüzü içeriği bakımından insana ait ad – soyad gibi bilgileri örneğin çalışanlarla da ilgili yapıda kullanılabilir. Bu açıdan arayüzü aşağıdaki şekilde düzenlemek daha yararlı olabilir.

```
interface IInsan
    {
        string Ad { get; }
        string Soyad { get; set; }
    }

interface IMusteri : IInsan
    {
        string Id { get; }
        void EkranaYazdir();
    }
```

Dikkat edilirse kalıtım alan IMusteri arayüzü IInsan arayüzü üyelerini tekrar yazmak zorunda değildir. IMusteri arayüzü kalıtım verdiği sınıflara otomatik olarak kalıtım almış olduğu arayüzlerin üyelerini iletir.

```
class Fatura : IMusteri, ISiparis
    {

        private string _id;
        public string Id
        {
            get { return _id; }
        }

        private string _adi;
        public string Ad
        {
            get { return _adi; }
            set { _adi = value; }
        }

        private string _soyad;
        public string Soyad
        {
            get
            {
                return _soyad;
            }
            set
```

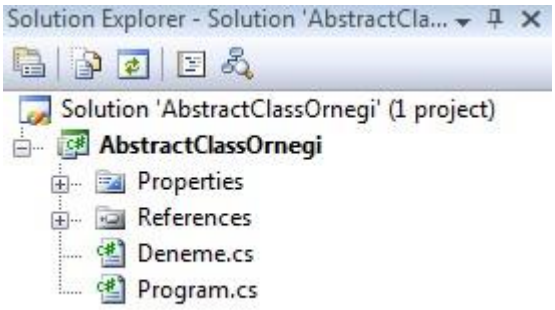
```
            {
                _soyad = value;
            }
        }

        private int _siparisId;
        public int SiparisID
        {
            get
            {
                return _siparisId;
            }
            set
            {
                _siparisId = value;
            }
        }

        private string _urunAdi;
        public string UrunAdi
        {
            get
            {
                return _urunAdi;
            }
            set
            {
                _urunAdi = value;
            }
        }

        private double _fiyat;
        public double Fiyat
        {
            get
            {
                return _fiyat;
            }
            set
            {
                _fiyat = value;
            }
        }

        private int _miktar;
        public int Miktar
        {
            get
```

```
            {
                return _miktar;
            }
            set
            {
                _miktar = value;
            }
        }

        // Hem IMusteri hem de ISepet arayüzlerinden EkranaYazdir() Metodu
gelmesi rağmen
        // sadece bir defa yazılmıştır.
        // Her iki arayüzden gelen EkranaYazdır metotları kullanılmak
isteniyorsa önce arayüzün
        // ismi saha sonra  ve metudun adı çağrılarak yazılmalıdır.
        // Erişim belirteçleri kaldırılmalıdır.

        //public void EkranaYazdir()
        //{
        //
        //}

        void IMusteri.EkranaYazdir()
        {
            Console.WriteLine(" Adı: {0} Soyadı: {1} Müşteri Id: {2}",
this.Ad, this.Soyad, this.Id);
        }

        void ISiparis.EkranaYazdir()
        {
            Console.WriteLine("Ürün adı: {0} Ürün Miktarı: {1} Ürün Fiyatı:
{2} Sipariş Numarası:{3}", this.UrunAdi, this.Miktar, this.Fiyat,
this.SiparisID);
        }
    }
```

8.2. Abstract (Soyut) Sınıflar

Birbirleriyle benzer işlev ve özelliklere sahip sınıfların kodlarını tekrar tekrar yazmaktansa kalıtım kullanarak bu ortak kodları tek bir sınıf altında toplayabiliriz. Bununla beraber eğer ki bu ortak kodları üzerinde toplayan sınıfın yeni bir örneği (instance) oluşturulmasına izin verilmek istenmiyorsa abstract anahtar sözcüğünü ile bunu engelleyebiliriz.

Yeni bir proje açalım ve - şekilde de görülebileceği gibi- sınıfımızın adını Deneme olarak belirtelim.




```
abstract class Deneme
    {

    }
```

Abstract Class'larin new anahtar sözcülüğü ile yeni bir örneği oluşturulamazlar.

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

class AbstractClassOrnegi.Deneme

Error:
Cannot create an instance of the abstract class or interface 'AbstractClassOrnegi.Deneme'

Not: sealed anahtar sözcüğü kalıtım vermesini engeller.

abstract sınıflar başka sınıflara temel sınıf olmak görevini üstlendiklerinden dolayı tek başlarına bulunmaları anlamlı değildir. Temel sınıf görevinde olmaları dolayısıyla;

1. New anahtar sözcülü ile yeni bir örneği oluşturulamayan sınıflardır.
2. Her ne kadar yeni bir örneği oluşturulmasa da kalıtım veren diğer sınıfların tüm özelliklerini taşırlar.
3. Kalıtım verdikleri sınıfları bazı durumlarda yönlendirmeleri gerekebilir (mecbur bırakmaları).

Kendisinden kalıtım alan sınıflarda bu mecburiyetleri deklare ederek ezmek zorundadırlar.

8.2.1. Metot ve Özelliklerin Abstract Olması

C#'ta abstract metotlar ve abstract property ler olarak adlandıracağımız ve alt sınıflarda yazılması mecburi olan iş mantıkları ancak ve ancak abstract sınıflar içersinde olabilirler ve public veya protected olmak zorundadırlar.

Örnek:

Bir mağazada satılan ürünlerin adlarını ve alış fiyatlarını tutabileceğimiz bir programda; kitap, yiyecek ve giysi reyonlarının olduğunu bunların değişik KDV ve kar oranları ile satış yapılacağını düşünelim. Bu durumda;

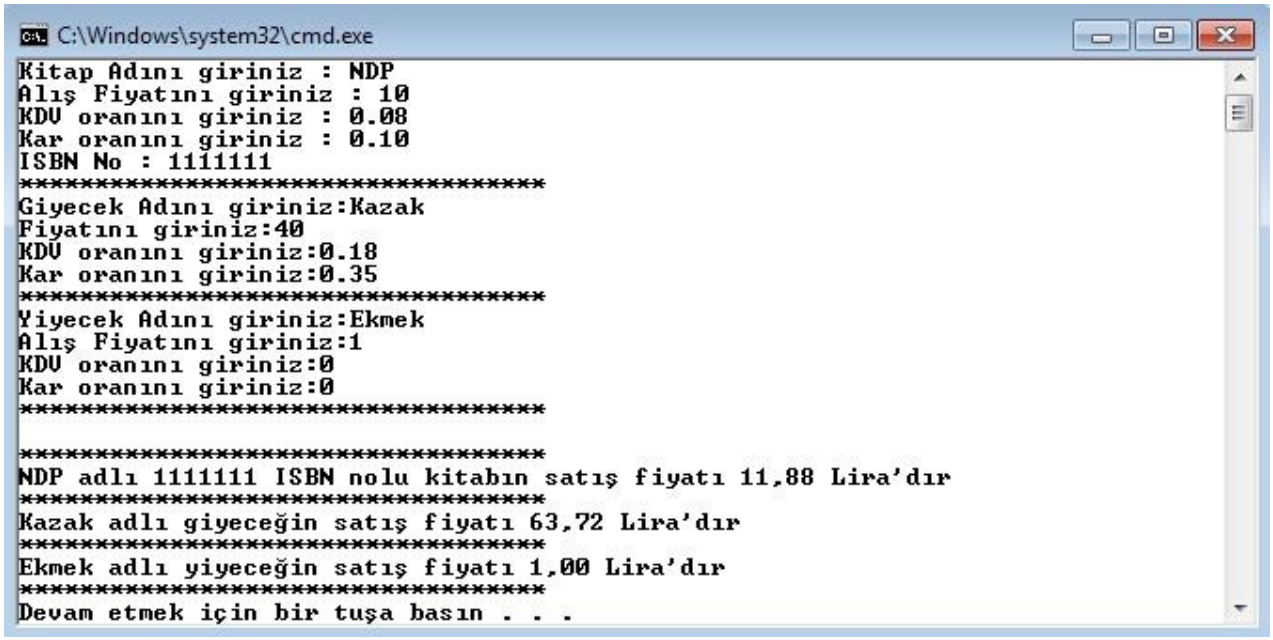
```
abstract class Urun
    {
        public abstract string UrunAdi
        {
            get; set;
        }

        private double m_AlisFiyati;
        public double AlisFiyati
        {
            get { return m_AlisFiyati; }
            set { m_AlisFiyati = value; }
        }

        private double m_KarOrani;
```

```
        public double KarOrani
        {
            get { return m_KarOrani; }
            set { m_KarOrani = value; }
        }

        private double m_KDV;
        public double KDV
        {
            get { return m_KDV; }
            set { m_KDV = value; }
        }

        public virtual double satisFiyatiHesapla()
        {
            return (AlisFiyati * ((1 + KarOrani / 100)) * (1 + KDV / 100));
        }

        public abstract void ekranaYadir();

    }

    class Yiyecek : Urun
    {

        private string m_YiyecekAdi;
        public override string UrunAdi
        {
            get
            {
                return m_YiyecekAdi;
            }
            set
            {
                m_YiyecekAdi = value;
            }
        }

        public override void ekranaYadir()
        {
            Console.WriteLine("{0} adlı yiyeceğin satış fiyatı {1:F2}
Lira'dır", this.m_YiyecekAdi, this.satisFiyatiHesapla());
            Console.WriteLine("**********************************");
        }
    }

    class Giyecek : Urun
    {
```

```
        private string m_giyecekAdi;
        public override string UrunAdi
        {
            get
            {
                return m_giyecekAdi;
            }
            set
            {
                m_giyecekAdi = value;
            }
        }
        public override void ekranaYadir()
        {
            Console.WriteLine("{0} adlı giyeceğin satış fiyatı {1:F2}
Lira'dır", this.m_giyecekAdi, this.satisFiyatiHesapla());
            Console.WriteLine("**********************************");
        }
    }

    class Kitap : Urun
    {
        private string m_kitapAdi;
        public override string UrunAdi
        {
            get
            {
                return m_kitapAdi;
            }
            set
            {
                m_kitapAdi = value;
            }
        }

        private string m_ISBNNo;
        public string ISBNNo
        {
            get { return m_ISBNNo; }
            set { m_ISBNNo = value; }
        }

        public override void ekranaYadir()
        {
            Console.WriteLine("{0} adlı {1} ISBN nolu kitabın satış fiyatı
{2:F2} Lira'dır", this.m_kitapAdi, this.ISBNNo, this.satisFiyatiHesapla());
            Console.WriteLine("**********************************");
        }
```

```
    }

    class Program
    {
        static void Main(string[] args)
        {
            //Kitap Bilgileri
            Kitap yeniKitap = new Kitap();

            Console.Write("Kitap Adını giriniz : ");
            yeniKitap.UrunAdi = Console.ReadLine();
            Console.Write("Alış Fiyatını giriniz : ");
            yeniKitap.AlisFiyati = Convert.ToDouble(Console.ReadLine());
            Console.Write("KDV oranını giriniz : ");
            yeniKitap.KDV = Convert.ToDouble(Console.ReadLine());
            Console.Write("Kar oranını giriniz : ");
            yeniKitap.KarOrani = Convert.ToDouble(Console.ReadLine());
            Console.Write("ISBN No : ");
            yeniKitap.ISBNNo = Console.ReadLine();

            Console.WriteLine("**********************************");
            // Giyecek Bilgileri
            Giyecek yeniGiyecek = new Giyecek();

            Console.Write("Giyecek Adını giriniz:");
            yeniGiyecek.UrunAdi = Console.ReadLine();
            Console.Write("Fiyatını giriniz:");
            yeniGiyecek.AlisFiyati = Convert.ToDouble(Console.ReadLine());
            Console.Write("KDV oranını giriniz:");
            yeniGiyecek.KDV = Convert.ToDouble(Console.ReadLine());
            Console.Write("Kar oranını giriniz:");
            yeniGiyecek.KarOrani = Convert.ToDouble(Console.ReadLine());

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

            //Yiyecek Bilgileri
            Yiyecek yeniYiyecek = new Yiyecek();

            Console.Write("Yiyecek Adını giriniz:");
            yeniYiyecek.UrunAdi = Console.ReadLine();
            Console.Write("Alış Fiyatını giriniz:");
            yeniYiyecek.AlisFiyati = Convert.ToDouble(Console.ReadLine());
            Console.Write("KDV oranını giriniz:");
            yeniYiyecek.KDV = Convert.ToDouble(Console.ReadLine());
            Console.Write("Kar oranını giriniz:");
            yeniYiyecek.KarOrani = Convert.ToDouble(Console.ReadLine());

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

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

            //Ekrana Yazdırma
            yeniKitap.ekranaYadir();
            yeniGiyecek.ekranaYadir();
            yeniYiyecek.ekranaYadir();

        }
    }
```



Örnek Uygulama: Alan hesapla isimli uygulamanın son hali;

```
abstract class SekilBase
    {
        private double r;
        protected double R
        {
            get
            {
                return r;
            }
            set
            {
                if (value < 0)
                    r = 0;
                else
                    r = value;
            }
        }

        private double h;
        protected double H
```

```
        {
            get
            {
                return h;
            }
            set
            {
                if (value < 0)
                {
                    h = 0; r = 0;
                }
                else
                    h = value;
            }
        }

        public SekilBase() { }

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

        protected abstract double AlanHesapla();

        public void EkranaYazdir()
        {
            Console.WriteLine(this.GetType().Name + " Alanı    = {0:F2}",
this.AlanHesapla());
        }
    }

    class Nokta : SekilBase
    {
        protected override double AlanHesapla()
        {
            return 0;
        }
    }

    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;
        }
```

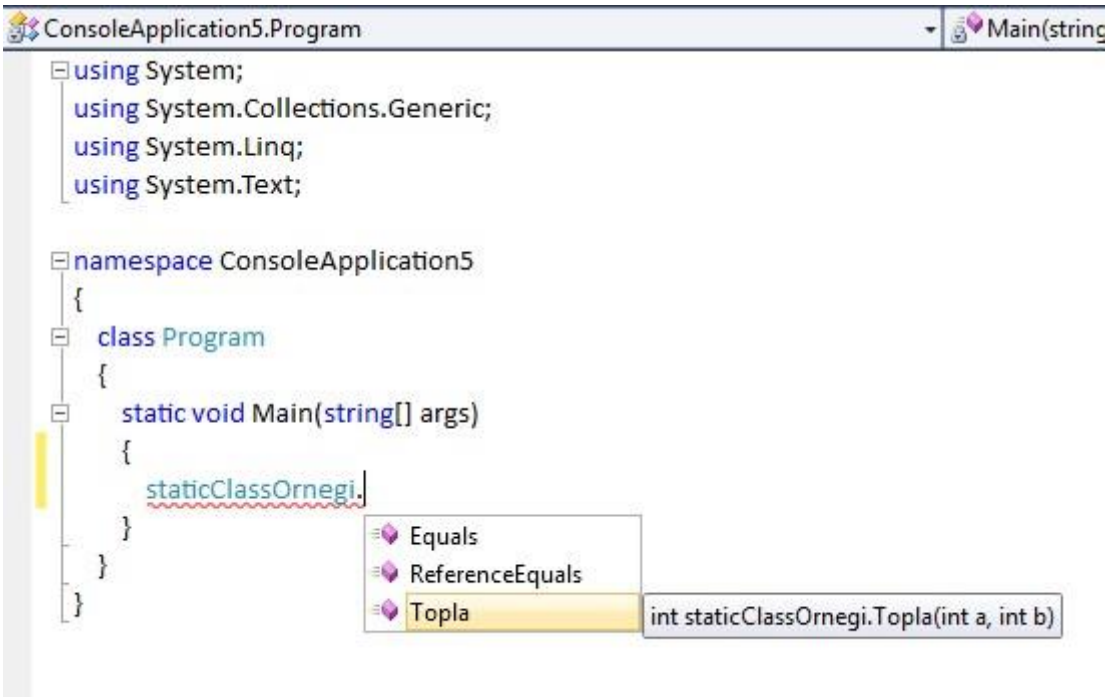
```
    }

    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 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("Yukselik: ");
            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();
        }
    }
```

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

8.3. Static Kavramı

Bir sınıfın herhangi üyesine sınıfın yeni bir örneğini oluşturmadan erişmek istenildiği taktirde static anahtar sözcüğü devreye girecektir.

```
class staticClassOrnegi
    {
        public static int Topla(int a, int b)
        {
            return a + b;
        }
    }
```



Örneğimizde staticClassOrnegi adındaki sınıfımızda public static int Topla(int a, int b) şeklinde static bir metot tanımlanmıştır. Main içerisinde ise new anahtar sözcüğü kullanılarak yeni bir örneği oluşturulmadan bu metoda uluşabilmemizin nedeni static anahtar sözcüğüdür.

Main metotumuzu aşağıdaki gibi düzenlersek sonuç olarak ekrana 13 yazacaktır.

```
class Program
    {
        static void Main(string[] args)
        {
           Console.WriteLine( staticClassOrnegi.Topla(5, 8));
        }
    }
```

Benzeri şekilde metotlar static olabileceği gibi değişkenlerimiz de static olabilir.

```
class MyClass
    {
        public string adi;
        public double kdv;

        public static int fiyat;
    }
```

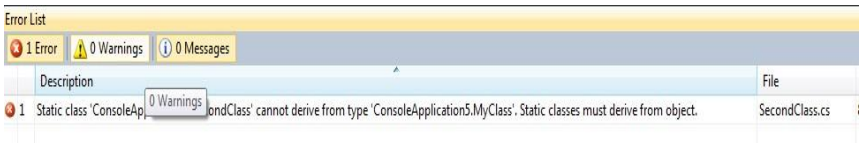
```
class Program
    {
        static void Main(string[] args)
        {
            MyClass.fiyat = 2300;

            Console.WriteLine(MyClass.fiyat);

            Console.WriteLine();

        }
    }
```

static değişkenlere değer atanabilirken cons ile tanımladığımız değişkenlere ancak oluşturulurken atanabilir ve bir daha değiştirilemezler. Benzeri şekilde readonly açıkça belirtilmediği takdirde static değildir.

8.3.1. Sınıfların static olması

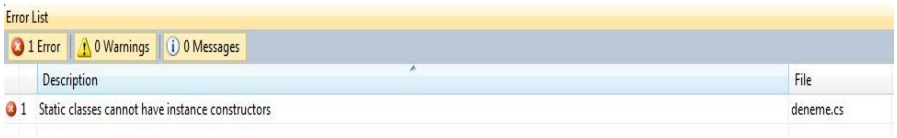
- static olarak tanımlanmış sınıflara ait nesne örneklerini oluşturulamaz.

```
static class MyClass
    {

    }

    class Program
    {
        static void Main(string[] args)
        {

            MyClass mc = new MyClass();

        }
    }
```



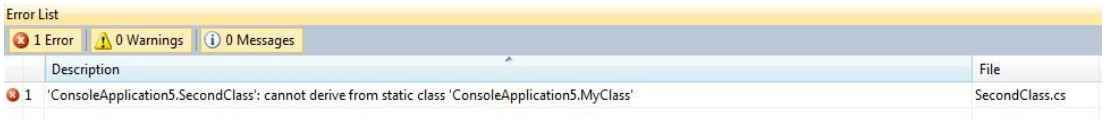
- static olarak tanımlanmış sınıfların kurucuları ve yıkıcıları olmaz.

```
static class Deneme
    {
        public Deneme()
        {
            Console.WriteLine("dfsdf");
        }
    }
```



- static olarak tanımlanmış sınıflardan kalıtım alınamaz.

```
static class MyClass
  {

      public string adi;

      public static string a = "Merhaba";

      public static int Topla(int a, int b)
      {
          return a + b;
      }

  }

  class SecondClass : MyClass
  {

  }
```



- static olarak tanımlanmış sınıflar kalıtım vermez.

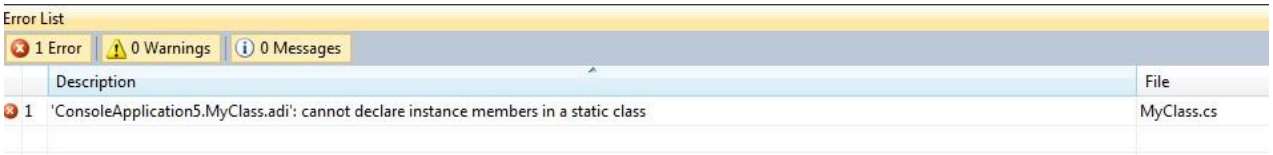
```
class BaseClass
    {

    }

    static class MyClass : BaseClass
    {

    }
```

- static olarak tanımlanmış sınıflardan başka static sınıflar da türetilemez.

```
static class MyClass
    {

        public string adi;

        public static string a = "Merhaba";

        public static int Topla(int a, int b)
        {
            return a + b;
        }

    }

    static class SecondClass : MyClass
    {
        // Hata !! Sınıf static olsa dahi static bir sınıftan kalıtım
alınamaz.
    }
```

- static sınıflar içerisinde statik olmayan üyeler oluşturulamaz.

```
static class MyClass
    {

        public string adi;

        public static string a = "Merhaba";

        public static int Topla(int a, int b)
        {
            return a + b;
        }

    }
```



sealed anahtar sözcülüğü ile kalıtım alınması yasaklanan sınıflar kurucularının public olması durumunda yeni bir örneği oluşturulabilir. Eğer daha kuralcı bir kalıtım yasaklanması isteniyorsa static kavramı kullanılabilir. (static classların kalıtım almadığı da unutulmamalıdır.)

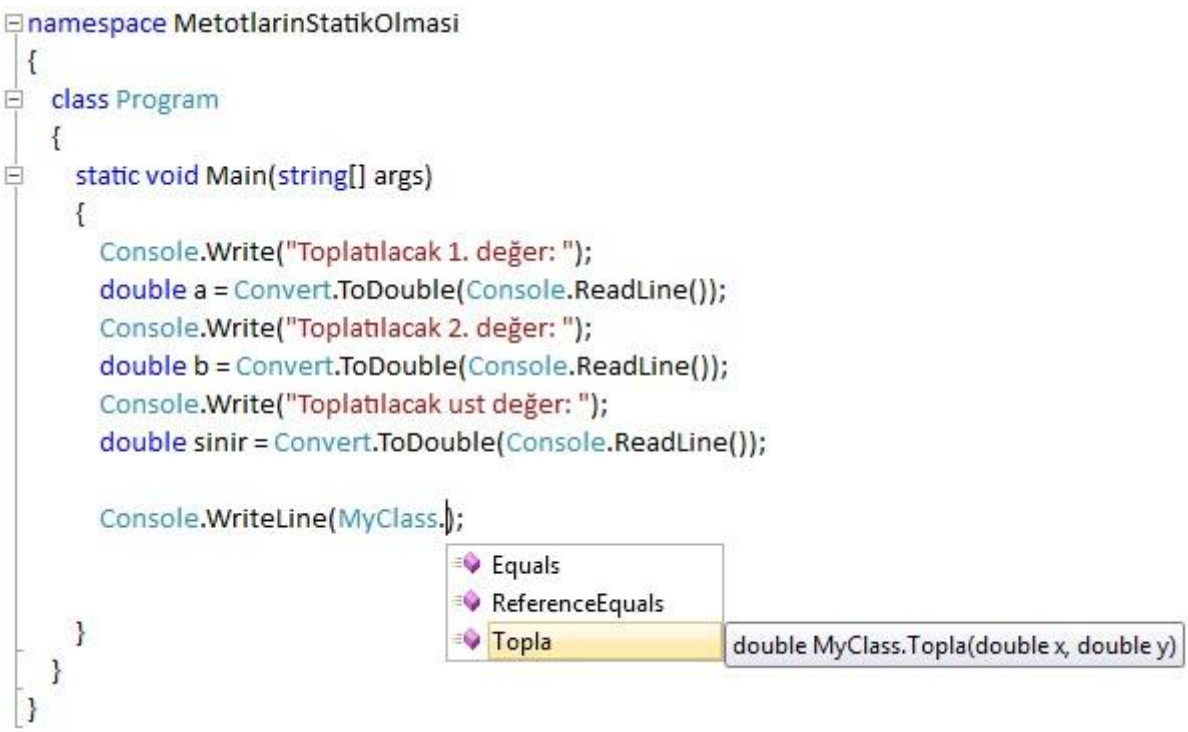
```
public class TemelAritmetik
    {
        public static double Toplam(double deger1, double deger2)
        {
            return deger1 + deger2;
        }
    }

    public class AltAritmetik : TemelAritmetik
    {

    }

    class Program
    {
        static void Main(string[] args)
        {
            AltAritmetik altAritmetik = new AltAritmetik();
            altAritmetik.AltIslemler();
            double sonuc = AltAritmetik.Toplam(1, 2);

        }
    }
```

Bu durumu engellemek için sınıfın static yapılması gerekir.

8.3.3. Metotların static olması

- Bir sınıf içerisinde static olmayan metotlar static olan bir metodu kullanabilir.

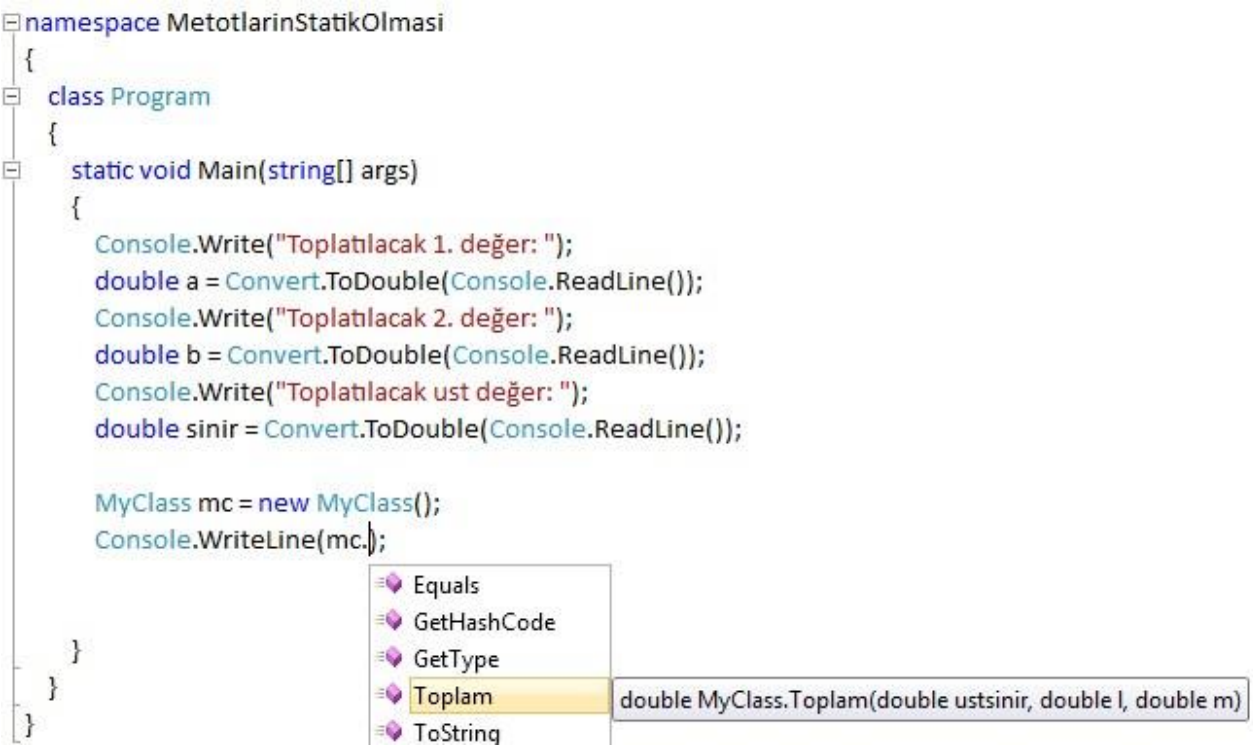
```
class MyClass
    {
        public static double Topla(double x, double y)
        {
            return x + y;
        }

        public double Toplam(double ustsinir, double l, double m)
        {
            double toplam = 0;

            for (double i = 1; i <= ustsinir; i++) toplam += Topla(l, m);

            return toplam;
        }
    }
```



```
class Program
    {
        static void Main(string[] args)
        {
            Console.Write("Toplatılacak 1. değer: ");
            double a = Convert.ToDouble(Console.ReadLine());
            Console.Write("Toplatılacak 2. değer: ");
            double b = Convert.ToDouble(Console.ReadLine());
            Console.Write("Toplatılacak ust değer: ");
            double sinir = Convert.ToDouble(Console.ReadLine());

            Console.WriteLine(MyClass.Topla(a, b));
        }
    }
```

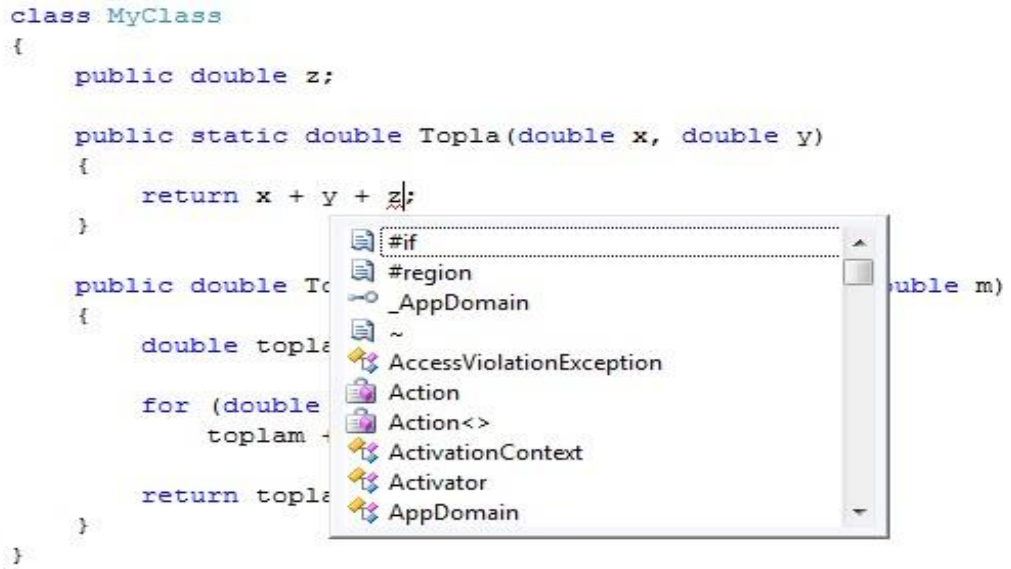
MyClass isimli sınıftan yeni bir örnek oluşturmadığımız için sadece Topla() isimli static metodu görebilmekteyiz. Ancak bize Toplam() isimli metot gerekli. Dolayısıyla yeni bir örnek oluşturmak zorundayız.

```
class Program
    {
        static void Main(string[] args)
        {
            Console.Write("Toplatılacak 1. değer: ");
            double a = Convert.ToDouble(Console.ReadLine());
            Console.Write("Toplatılacak 2. değer: ");
            double b = Convert.ToDouble(Console.ReadLine());
            Console.Write("Toplatılacak ust değer: ");
            double sinir = Convert.ToDouble(Console.ReadLine());

            MyClass mc = new MyClass();
            Console.WriteLine(mc.Toplam(a, b, sinir));

        }
    }
```



Görüldüğü gibi MyClass sınıfından yeniClass isimli yeni bir örnek(nesne) oluşturduğumuzda artık Toplam() isimli metodu görebilirken, static olarak tanımladığımız Topla() metodu görünmemektedir.

- Bir sınıf içersinde static olan üyeler static olmayan üyelere erişemezler.



```
class MyClass
{
    public double z;

    public static double Topla(double x, double y)
    {
        ekranaYazdir();
        void MyClass.ekranaYazdir()

        Error:
          An object reference is required for the non-static field, method, or property 'ConsoleApplication2.MyClass.ekranaYazdir()'
    }
    public void ekranaYazdir()
    {
        Console.WriteLine("Ekrana Yaz");
    }

    public double Toplam(double ustsinir, double l, double m)
    {
        double toplam = 0;

        for (double i = 1; i <= ustsinir; i++)
            toplam += Topla(l, m);

        return toplam;
    }


}
```

Static metotlar içinde static olmayan değişken yada metotlar kullanılmazlar.

8.3.4. Static Sınıf Kurucuları.

```
static class MyStaticClass
{

    public MyStaticClass()
    {        Static classes cannot have instance constructors

    }
}
```

Static sınıflar kurucu bulunduramazlar. Çeşitli şekillerde kurucular aşırı yüklenseler(overload) dahi sonuç değişmeyecektir.

Sonuç olarak eğer bir sınıf static ise ancak ve ancak static üyeler bulundurabilirler.