5.4 Yapıcı Fonksiyonlar ve Kalıtım

C#'ta yapıcı metot kullanımında bazı kurallar vardır:

- Yapıcı metotlar fiziksel olarak türetilmez.
- Yavru sınıf türünden bir nesne yaratıldığında önce ana sınıfın parametre almayan yapıcı metodu, ardından yavru sınıftaki imzaya uyan yapıcı metot çalıştırılır.
- Yavru sınıf türünden nesne yaratımında daima yavru sınıfın imzaya uyan bir yapıcı metodu olması gerekir.
- Yavru sınıf türünden nesne yaratımlarında, ana sınıfın parametre almayan yapıcı metodu yavru sınıfın üye elemanlarıyla işlem yapar.
- Yavru sınıf türünden nesne yaratımında, yavru sınıftaki ilgili (imzası uygun) yapıcı metoda base takısı eklenmişse ana sınıfın parametre almayan yapıcı metodu çalıştırılmaz.

Örnek:

Aşağıdaki programda önce ana sınıf yapıcısındaki mesaj sonra alt sınıf yapıcısındaki mesaj ekrana basılır.

```
using System;
namespace Kalitim
{
    class Sayi
    {
        public Sayi()
        {
            Console.WriteLine("Ana sınıf yapıcısı çalıştı");
        }
    }
    class Deger : Sayi
    {
        public Deger(int n)
        {
            Console.WriteLine("Alt sınıf yapıcısı çalıştı. Parametre:"
     + n);
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            Deger y = new Deger(5);
            Console.ReadKey();
```

```
        }
    }
}
```

Aynı kodu aşağıdaki gibi yazarsak ne olur? (Alt sınıfa yapıcı fonksiyon yazmasak):

```
using System;
namespace Kalitim
{
    class Sayi
    {
        public Sayi()
        {
            Console.WriteLine("Ana sınıf yapıcısı çalıştı");
        }
    }
    class Deger : Sayi
    {
    }
    class Program
    {
        static void Main(string[] args)
        {
            Deger y = new Deger();
            Console.ReadKey();
        }
    }
}
```

Bir sınıfta yapıcı metot olmadığı durumlarda otomatik olarak parametresiz çalışan bir varsayılan yapıcı metot oluşturulur. Yukarıdaki kodda nesne olştururken parametre vermediğimiz için:

```
Deger y = new Deger();
```

kod düzgün çalışır.

Örnek:

Aşağıdaki örnekte ana sınıf yavru sınıfın elemanlarını kullnır. Ekran çıktısı 100 olur:

```
using System;
namespace Kalitim
{
```

```
    class Sayi
    {
        public int puan;
        public Sayi()
        {
            puan = 100;
        }
    }
    class Deger : Sayi
    {

    }
    class Program
    {
        static void Main(string[] args)
        {
            Deger y = new Deger();
            Console.WriteLine(y.puan);
            Console.ReadKey();
        }
    }
}
```

Örnek

Aşağıdaki örnekte ana sınıfın varsayılan (parametresiz) kurucusu olmadığı için kod hata üretir.

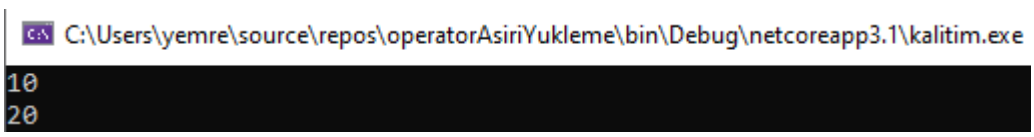
```
using System;
namespace Kalitim
{
    class Sayi
    {
        public Sayi(string prm)
        {
            //...
        }
    }
    class Deger : Sayi
    {

    }
    class Program
    {
        static void Main(string[] args)
        {
            Deger y = new Deger();
            Console.ReadKey();
        }
    }
```

```
}
```

Örnek:

Alt sınıfın yapıcı metoduna eklenen **base** takısı ile ana sınıfın bir yapıcı metodunu çalıştırabiliriz. Bu şekilde alt sınıf türünden bir nesne oluşturulduğunda ana sınıfın parametre almayan yapıcı metodu çalıştırılmayacaktır.

```
using System;
namespace Kalitim
{
    class Sayi
    {
        public int oge1;
        public Sayi()
        {
            Console.WriteLine("Parametresiz yapıcı çalıştı");
        }
        public Sayi(int prm1)
        {
            oge1 = prm1;
        }
    }
    class Deger : Sayi
    {
        public int oge2;
        public Deger(int prm1, int prm2):base(prm1)
        {
            oge2 = prm2;
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            Deger y = new Deger(10, 20);
            Console.WriteLine(y.oge1 + " \n"  + y.oge2);
            Console.ReadKey();
        }
    }
}
```



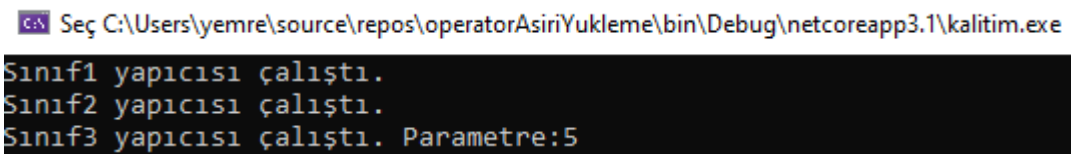
5.5 Çoklu Kalıtım

B sınıfı A sınıfından ve C sınıfı da B sınıfından türetilebilir. Bu durumda C sınıfı türünden bir nesne oluşturduğumuzda eğer C sınıfının ilgili yapıcı metoduna base takısını eklememişsek önce A, sonra B, sonra da C sınıfının yapıcı metotları çalıştırılır (anadan yavruya doğru).

Ayrıca C sınıfı hem A'nın hem de B'nin bütün üye elemanlarına sahip olur.

Örnek:

```
using System;
namespace Kalitim
{
    class Sinif1
    {
        public Sinif1()
        {
            Console.WriteLine("Sınıf1 yapıcısı çalıştı.");
        }
    }
    class Sinif2 : Sinif1
    {
        public Sinif2()
        {
            Console.WriteLine("Sınıf2 yapıcısı çalıştı.");
        }
    }
    class Sinif3 : Sinif2
    {
        public Sinif3(int n)
        {
            Console.WriteLine("Sınıf3 yapıcısı çalıştı. Parametre:" +
 n);
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            Sinif3 y = new Sinif3(5);
            Console.ReadKey();
        }
    }
}
```


Seç C:\Users\yemre\source\repos\operatorAsiriYukleme\bin\Debug\netcoreapp3.1\kalitim.exe
Sınıf1 yapıcısı çalıştı.
Sınıf2 yapıcısı çalıştı.
Sınıf3 yapıcısı çalıştı. Parametre:5