Kodun tekrar kullanılabilirliği ve merkezi yönetimi

AlanHesapla isimli örneğimize z isimli yeni bir `Yamuk` nesnesi ekleyelim.

```
// z isimli yamuk nesnesine değerler atama
z.taban = 6;
z.tavan = 7;
z.yukseklik = 8;

// z yamuğunun alanı hesaplanır
double alan3 = (z.taban + z.tavan) / 2 * z.yukseklik;

// z yamuğunu ekrana yazdırır
Console.WriteLine("Z: " + alan3);
```

Örnekler çoğaltıldıkça aynı kodların (alan hesaplama ve ekrana yazdırma kodları) defalarca tekrarlandığı görülecektir. Bu durum NDP'nin temel hedeflerinden biri olan kodun tekrar kullanılabilirliği ile çelişmektedir.

Ekrana yazdırma kodlarımız olan `Console.WriteLine("Z: " + alan3)` satırında sonuç olarak ekrana çıkarttığı şekilde değil de "Yamuğun Alanı … metre karedir." şeklinde bir gösterim istenilirse, her yamuk için ekrana yazdırma kodlarını teker teker değiştirilmesi gerekecektir. Zor, dikkat gerektiren bu durum aslında NDP'nin prensiplerinden "Her sınıfın tek görevi/sorumluluğu olmalıdır" prensibine de aykırıdır. Çünkü yamuğun alanını ekrana yazdıran kodlar `Program.cs` sınıfı içerisinde yer almaktadır. `Program.cs` sınıfının görevi yamuğun alanın ekrana yazdırmak değil programı işletmektir.

Benzeri şekilde yamuğun alanını hesaplamak `Program.cs` sınıfının sorumluluğunda olmaması gerekir. Dolayısıyla ekrana yazdırma ve alan hesaplama kodlarımız `Yamuk.cs` sınıfının içerisinde olmalıdır. Yeni kodlarımızı şöyle olmalıdır:

```
public class Yamuk
    {
        public double taban;
        public double tavan;
```

```
        public double yukseklik;

        public double AlanHesapla()
        {
            return  (taban + tavan) / 2 * yukseklik;
        }
        public void EkranaYazdir()
        {
            Console.WriteLine("Yamuğun alanı " + AlanHesapla() +
" metre karedir.");

        }
    }
class Program
    {
        static void Main(string[] args)
        {
            // yamuk isimli yamuk nesnesini oluşturma
            Yamuk yamuk = new Yamuk();
            // yamuk isimli yamuk nesnesine değerler atama
            yamuk.taban = 5;
            yamuk.tavan = 6;
            yamuk.yukseklik = 8;
            // yamuk yamuğunu ekrana yazdırır
            yamuk.EkranaYazdir();

            // x isimli yamuk nesnesini oluşturma
            Yamuk x = new Yamuk();
            // x isimli yamuk nesnesine değerler atama
            x.taban = 3;
            x.tavan = 4;
            x.yukseklik = 5;
            // x yamuğunu ekrana yazdırır
            x.EkranaYazdir();

            // z isimli yamuk nesnesini oluşturma
            Yamuk z = new Yamuk();
            // z isimli yamuk nesnesine değerler atama
            z.taban = 6;
            z.tavan = 7;
            z.yukseklik = 8;
            // z yamuğunu ekrana yazdırır
            z.EkranaYazdir();
        }
    }
```

Program.cs sınıfını incelediğimizde Yamuk.cs sınıfında public olarak tanımlanan AlanHesapla() metodunun kullanılmasına gerek olamadığını görmekteyiz. Bu durumda NDP nin ilk yapı taşı olan Sarmalama gerçeklenmemiş olmaktadır. Bu metodun private düzeye çekilmesi gerekmektedir. Bu durumda Yamuk.cs kodları aşağıdaki gibi olacaktır.

```
public class Yamuk
    {
        public double taban;
        public double tavan;
        public double yukseklik;

        private double AlanHesapla()
        {
            return  (taban + tavan) / 2 * yukseklik;
        }

        public void EkranaYazdir()
        {
            Console.WriteLine("Yamuğun alanı " + AlanHesapla() +
" metre karedir.");
        }

    }
```

EkranaYazdir() metodundaki ekrana yazdırma kodunu değiştirdiğimiz anda istenilen şekilde sonuç üretilecektir. Böylelikle NDP prensiplerinden "Değişime kapalı gelişime açık" prensibini de gerçeklemiş olmaktayız.

Kodun tekrar kullanılabilirliği ve kodun merkezi yönetimi böylelikle sağlanmış oldu. Sistemimize yeni geometrik şekiller ekleyebiliriz: Kare, dikdörtgen, daire, üçgen vb.