3.3 Metot (Yordam) Tanımlama

```
class oyuncu
    {
        public string ad;
        public string mevki;
        private int savunma;
        private int atak;
        private int topSurme;
        private double deger;

        public void calis(string konu)
        {
            if (konu == "savunma") savunma += 1;
            if (konu == "atak") atak += 1;
            if (konu == "topSurme") savunma += 1;
            degerBelirle();
        }
        public void degerBelirle()
        {
            deger *= 1.1;
        }
     }
```

Yukarıdaki kodda; `savunma`, `atak`, `topSurme` ve `değer` özellikleri `private` yapılmıştır. Artık nesneden ulaşılamaz. Metotlarla değiştirilebilir:

`calis` metodu futbolcunun çalışma alanına göre yetenek puanını artırır.

`degerBelirle` metodu futbolcunun güncel yeteneklerine göre yeni değerini belirler.

3.4 Soyutlama (Abstraction)

Karmaşıklığı azaltmak için kullanılır. Bir program çok sayıda sınıf yapısını içerebilir. Her sınıfın kendine ait özellikleri vardır. Soyutlamanın amacı her sınıfın özelliklerinin iyi bir şekilde belirlenmesidir. Örneğin oyuncu sınıfımız için `ad`, `mevki`, `savunma`, vs… özelliklerinin belirlenmesi işlemi bir soyutlamadır. Genel anlamda sınıfların kendine ait özelliklerinin belirlenmesi işlemine soyutlama denir.

```
class oyuncu
{
    public string ad;
    public string mevki;
    private int savunma;
```

```
    private int atak;
    private int topSurme;
    private double deger;
}
```

3.5 Saklama, Paketleme, Kapsülleme (Encapsulation)

Saklama, soyutlamayı desteklemek ya da güçlendirmek için bir sınıfın içyapısının gizlenmesidir. Saklama sayesinde özelliklere ve yordamlara ulaşılması engellenebilir. Sınıf içindeki özellikler private olarak tanımlanır ve bu özelliklere sadece yordamlar üzerinden ulaşabilmeye izin verilir.

```
using System;
namespace yEmre
{
    class oyuncu
    {
        private string ad;
        private string mevki;
        private int savunma;
        private int atak;
        private int topSurme;
        private double deger;

        public void setAd(string adi)
        {
            ad = adi;
        }
        public void setMevki(string mevkisi)
        {
            mevki = mevkisi;
        }
        public void setYetenek(int sav, int atk, int ts)
        {
            savunma = sav;
            atak = atk;
            topSurme = ts;
            degerBelirle();
        }
        public void calis(string konu)
        {
            if (konu == "savunma") savunma += 1;
            if (konu == "atak") atak += 1;
            if (konu == "topSurme") savunma += 1;
            degerBelirle();
        }
        public void degerBelirle()
        {
```

```
            deger = savunma * 0.3 + atak * 0.4 + topSurme * 0.5;
        }

        public double getDeger()
        {
            return deger;
        }
    }

    class Futbol
    {
        static void Main(string[] args)
        {
            oyuncu oyuncu1 = new oyuncu(); //1. nesne üretiliyor
            oyuncu oyuncu2 = new oyuncu(); //2. nesne üretiliyor

            oyuncu1.setAd("Emir");
            oyuncu1.setMevki("Santrafor");
            oyuncu1.setYetenek(45, 79, 81);

            oyuncu1.setAd("Kerem");
            oyuncu1.setMevki("Stoper");
            oyuncu1.setYetenek(85, 59, 55);

            Console.WriteLine("Oyuncu degeri:" + oyuncu1.getDeger() +
    " M$");
            oyuncu1.calis("atak");
            Console.WriteLine("Oyuncu degeri:" + oyuncu1.getDeger() +
    " M$");

            Console.ReadKey();
        }
    }
}
```

`Set` ile başlayan yordamlar `void` yordam iken `get` ile başlayan yordamlar değer döndürdükleri için `return` içerir. Ve uygun `private` özelliği `return` eder.

3.6 Yapıcı (Constructor)

Yapıcılar ile: sınıftan bir nesne oluşturduğumuzda nesne özelliklerini oluşturma aşamasında belirleyebiliriz.

Yapıcı; sınıf ile aynı isimde olur ve bir değer döndürmez.

Yapıcı; dışarıdan değer alabilir.

```
using System;
using System.Collections;

namespace yEmre
{
    class oyuncu
    {
        private string ad;
        private string mevki;
        private int savunma;
        private int atak;
        private int topSurme;
        private double deger;

        public oyuncu(string adi, string mevkisi)
        {
            ad = adi;
            mevki = mevkisi;
        }


        public void setYetenek(int sav, int atk, int ts)
        {
            savunma = sav;
            atak = atk;
            topSurme = ts;
            degerBelirle();
        }

        #region metotlar
        #endregion
    }

    class Futbol
    {
        static void Main(string[] args)
        {
            oyuncu oyuncu1 = new oyuncu("Emir", "Santrafor");
            oyuncu oyuncu2 = new oyuncu("Kerem", "Stoper");
            oyuncu1.setYetenek(45, 79, 81);
            oyuncu2.setYetenek(85, 59, 55);

            Console.WriteLine("Oyuncu degeri:" + oyuncu1.getDeger() +
     " M$");
            oyuncu1.calis("atak");
            Console.WriteLine("Oyuncu degeri:" + oyuncu1.getDeger() +
     " M$");

            Console.ReadKey();
```

```
        }
    }
}
```

3.7 Yıkıcı (Destructor)

Yıkıcı metot ta yapıcı gibi sınıf ile aynı isme sahip olur ancak (~) karakteri ile başlar.

```
~oyuncu()
{
     Console.Write("Nesne ömrünü tamamladı.");
}
```

Sınıf bellekten çıkarılırken (sınıfın son durumunu kaydetme, log tutma, başka sınıfları tetikleme vb…) bazı işlemler yapmamız gerekirse bu işlemler yıkıcı metotta yapılabilir.

C++ gibi programlama dillerinde bir nesnenin bellekten silinmesi manuel bir süreç olarak yönetiliyordu. Java ve C#; bu metodun sadece gerektiğinde kullanılmasına imkân tanıyor. Destructor işlemi zorunlu değil. Garbage Collection (GC) (çöp toplama) mekanizması sayesinde işlemi biten nesneler bellekten otomatik olarak siliniyor.

```
GC.Collect();
```

Komutu ile çöp toplama çağrılabilir. Uygulama kapatıldığında bu işlem GC tarafından otomatik olarak yapılacaktır.

3.8 Özellik (Property) Metotları

Bir sınıf içerisindeki bazı alanlara her zaman ulaşılması gerekmez. Kapsülleme yöntemi ile bilinçsiz kullanım, veri kaybı ve güvenliği gibi sorunların önüne geçmeye çalışırız.

Aşağıdaki örnekte alanlar private tanımlandı. `setAtak` ve `getAtak` metotları değer belirlemek ve almak için yazıldı. `setAtak` metodu ile `private atak` alanının değeri `if` kontrolü ile denetlenerek girildi:

```
class oyuncu
{
    private string ad;
    private string mevki;
    private int atak;

    public oyuncu(string adi, string mevkisi)
    {
        ad = adi;
```

```
            mevki = mevkisi;
        }

        public void setAtak(int atk)
        {
            if (atak < 25)
                atak = 25;
            else
                atak = atk;
            degerBelirle();
        }

        public int getAtak()
        {
            return atak;
        }
    }
```

Yukarıdaki örnekte verilen nesne erişiminin yönetilmesi işlemini `property` (özellik) metotlarıyla da yapabiliriz.

`Property` metodu bir sınıfın içindeki alanları yöneten tek yordamdır (yukarıdaki örnekte 2 ayrı metot yazılmıştı).

Aynı yordamda hem değiştirme hem de belirli bir değişkeni döndürme işlemi yapılabilir. Bu işlemler `Property` metodu içinde `set` ve `get` bölümlerine yazılır.

```
using System;

namespace yEmre
{
    class oyuncu
    {
        private string ad;
        private string mevki;
        private int savunma;
        private int atak;
        private int topSurme;
        private double deger;

        public oyuncu(string adi, string mevkisi)
        {
            ad = adi;
            mevki = mevkisi;
        }

        public int Atak  //Burası property metodu. Parantez
    kullanılmaz
```

```
        {
            set{
                if (value < 25)
                    atak = 25;
                else
                    atak = value;
                degerBelirle();
            }
            get{
                return atak;
            }
        }
        public void setYetenek(int sav, int atk, int ts)
        {
            savunma = sav;
            atak = atk;
            topSurme = ts;
            degerBelirle();
        }
    }

    class Futbol
    {
        static void Main(string[] args)
        {
            oyuncu oyuncu1 = new oyuncu("Emir", "Santrafor");
            oyuncu1.setYetenek(45, 79, 81);
            Console.WriteLine("Oyuncu atak yeteneği:" + oyuncu1.Atak
    );

            Console.Write("Atak Değeri Girin:");
            int atk = int.Parse(Console.ReadLine());
            oyuncu1.Atak = atk;
            Console.WriteLine("Oyuncu atak yeteneği:" +
    oyuncu1.Atak);

            Console.ReadKey();
        }
    }
}
```

Karşılaştırma:

2 yordamlı kodlama	**Property metodu ile kodlama**

`class oyuncu`
`{`
`  private string ad;`
	`class oyuncu`
	`{`
	`  private string ad;`

```
    private string mevki;
    private int atak;

    public oyuncu(string adi, string
    mevkisi)
    {
        ad = adi;
        mevki = mevkisi;
    }

    public void setAtak(int atk)
    {
        if (atak < 25)
        atak = 25;
        else
              atak = atk;
              degerBelirle();


    public int getAtak()
    {
        return atak;
    }
      }
```

```
    private string mevki;
    private int atak;

    public oyuncu(string adi,
         string mevkisi)
    {
         ad = adi;
         mevki = mevkisi;
    }

    public int Atak  //Burası
         property metodu. Parantez
         kullanılmaz
    {
         set{
              if (value < 25)
                   atak = 25;
              else
                   atak = value;

              degerBelirle();
         }
         get{
              return atak;
         }

    }
```

```
Console.Write("Atak Değeri
  Girin:");
int atk = Console.Read();
oyuncu1.setAtak(atk);
Console.WriteLine("Oyuncu atak
  yeteneği:" + oyuncu1.getAtak());
```

```
Console.Write("Atak Değeri
         Girin:");
int atk = int.Parse(Console.ReadLine());
oyuncu1.Atak = atk;
Console.WriteLine("Oyuncu atak
         yeteneği:" +
         oyuncu1.Atak);
```

Bir property her zaman get veya set alanlarını içermek zorunda değildir. Sadece get alanı bulunduran bir property read-only sadece set alanı bulunduran bir property write-only'dir denir.