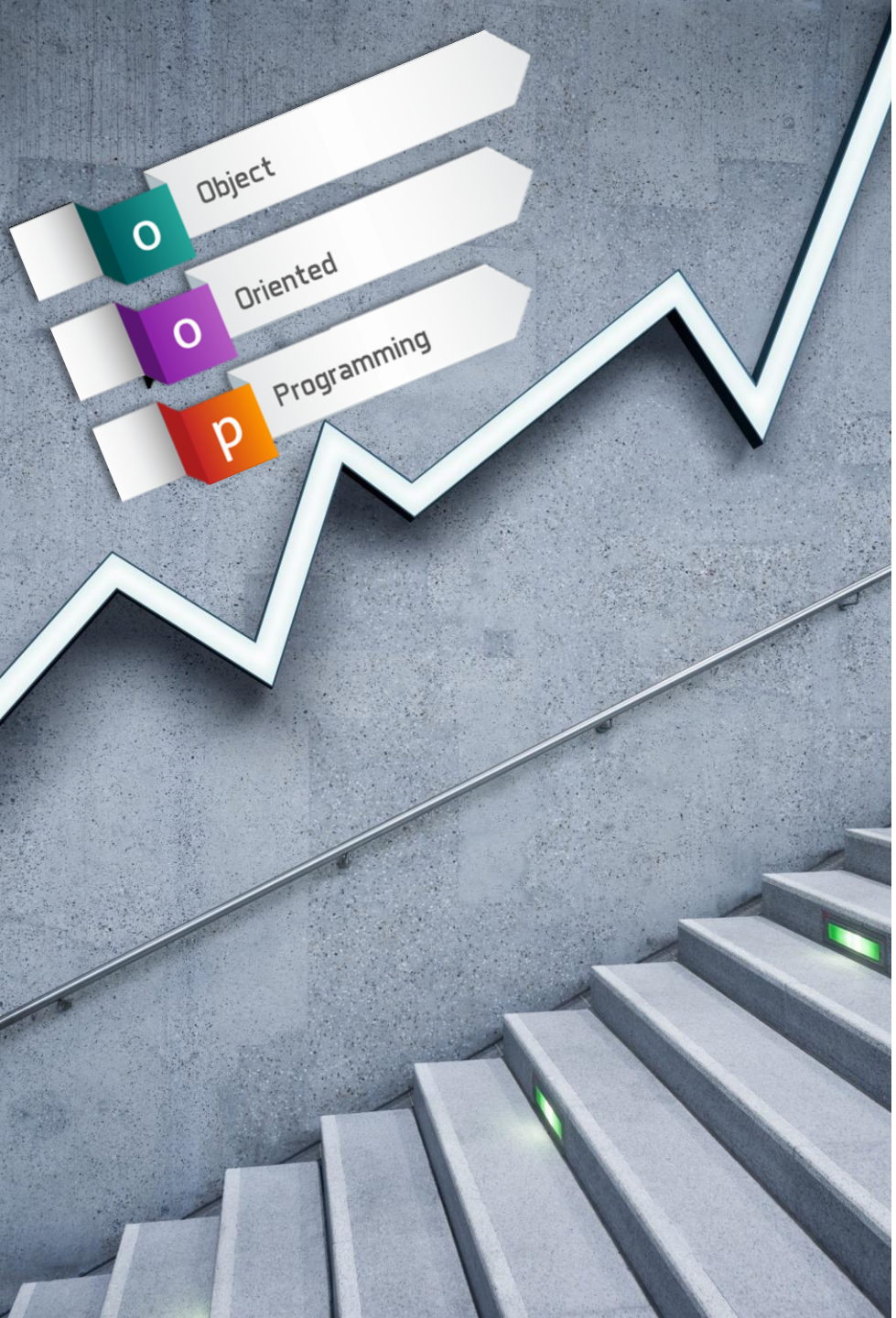
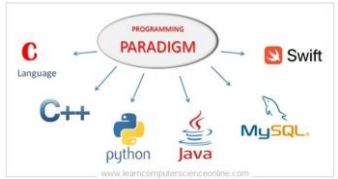




C#
Object Oriented Programming -OOP

this
ve
Constructor

ADEM AKKUŞ | Bilgisayar Mühendisi | Uzman Bilişim Teknolojileri Öğretmeni | Eğitmen

This Anahtar Sözcüğü

---

this anahtar sözcüğü sınıfın mevcut örneğine (referans eder) atıfta bulunur ve ayrıca bir extension method (uzantı metodunun) ilk parametresinin değiştiricisi olarak kullanılır.

This Anahtar Sözcüğü

this anahtar sözcüğünün bir extension method (uzantı metodunun) ilk parametresinin değiştiricisi olarak kullanılmasına örnek.

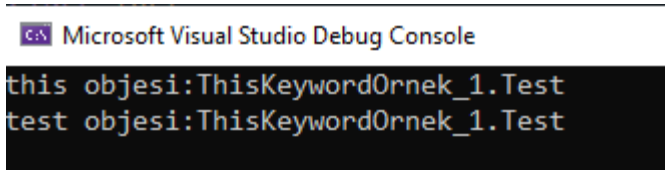
```
namespace ExtensionMethods
{
    public static class MyExtensions
    {
        public static int WordCount(this string str)
        {
            return str.Split(new char[] { ' ', '.', '?' },
                             StringSplitOptions.RemoveEmptyEntries).Length;
        }
    }
}
```

This Anahtar Sözcüğü

this anahtar sözcüğü sınıfın mevcut örneğine (referans eder) atıfta bulunmasına örnek.

```
namespace ThisKeywordOrnek_1
{
    3 references
    class Test
    {
        int numara;
        1 reference
        public Test(int numara)
        {
            //this.numara ifadesi int numara field'ini referans eder.
            this.numara = numara;
            Console.WriteLine("this objesi:"+this);
        }
    }
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            Test test = new Test(8);
            Console.WriteLine("test objesi:"+test);

        }
    }
}
```



This Anahtar Sözcüğü

this anahtar sözcüğü; Benzer adlara göre gizlenen üyeleri nitelemek için kullanılır, örneğin:

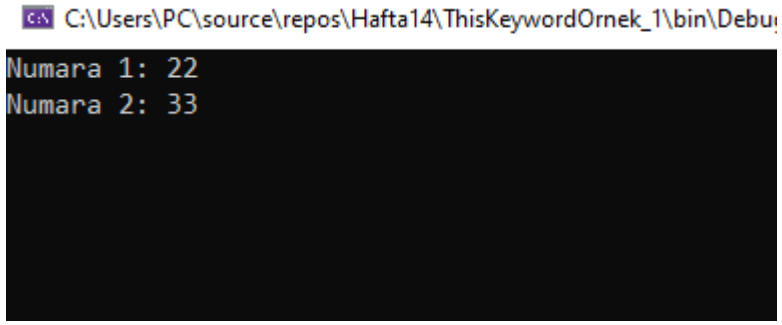
```
public class Calisan
{
    private string takmaAd;
    private string isim;

    0 references
    public Calisan(string isim, string takmaAd)
    {
        // Yapıcı metot parametreleri yerine sınıfın üyelerini nitelemek için bunu kullanın.
        this.takmaAd = takmaAd;    // this.takmaAd ifadesi takmaAd field'na referans eder
        this.isim = isim;          //this.isim ifadesi isim field'na referans eder
    }
}
```

This Anahtar Sözcüğü

this anahtar sözcüğü; Bir nesneyi parametre olarak diğer yöntemlere geçirmek için kullanılır, örneğin:

```
class Test
{
    int numara1;
    int numara2;
    1 reference
    public Test()
    {
        numara1 = 22;
        numara2 = 33;
    }
    1 reference
    public void ParametreAl(Test test1)
    {
        Console.WriteLine("Numara 1: " + numara1);
        Console.WriteLine("Numara 2: " + numara2);
    }
    1 reference
    public void Goster()
    {
        // this parametre olarak gönder
        ParametreAl(this);
    }
}
```



This Anahtar Sözcüğü

this anahtar sözcüğü; Dizin oluşturucuları (indexer) bildirmek için kullanılır, örneğin:

```
class Student
{
    private string[] isim = new string[6];
    // bir indexer tanımlama
    7 references
    public string this[int index]
    {
        // dizi elemanlarını değerini döndürme
        get
        {
            return isim[index];
        }
        // dizi elemanlarına değer atama
        set
        {
            isim[index] = value;
        }
    }
}
```

```
class Program
{
    0 references
    static void Main(string[] args)
    {
        Student student = new Student();
        student[0] = "Adem";
        student[1] = "Ebru";
        student[2] = "Kadir";
        student[3] = "Sultan";
        student[4] = "Mazlum";
        student[5] = "Mert";
        for (int i = 0; i < 6; i++)
        {
            Console.WriteLine(student[i] + " ");
        }
    }
}
```

Constructor (Yapıcı, İnşaedici,Kurucu)

Yapıcı metot, tüm sınıf veri üyelerini başlatmak için bir sınıfın nesnesi oluşturulduğunda otomatik olarak çağrılan C# 'daki özel metotlardır . İnşa edici, kurucu, yapıcı metot olarak Türkçe kaynaklarda geçmektedir.

Aynı sınıfın veri üyelerine başlangıç değerleri atamak için de kullanılır.

```
class personel
    {
        public personel() //Sınıf ismiyle aynı olduğuna dikkat
        {
            Console.WriteLine("Personel nesnesi oluşturuldu");
        }
    }
```

Constructor Özellikleri

- Sınıf ismi ile aynı olmalıdır.

```
class personel
    {
        public personel() //Sınıf ismiyle aynı olduğuna dikkat
        {
            Console.WriteLine("Personel nesnesi oluşturuldu");
        }
    }
```

- Erişim belirleyici **public** olmalıdır.

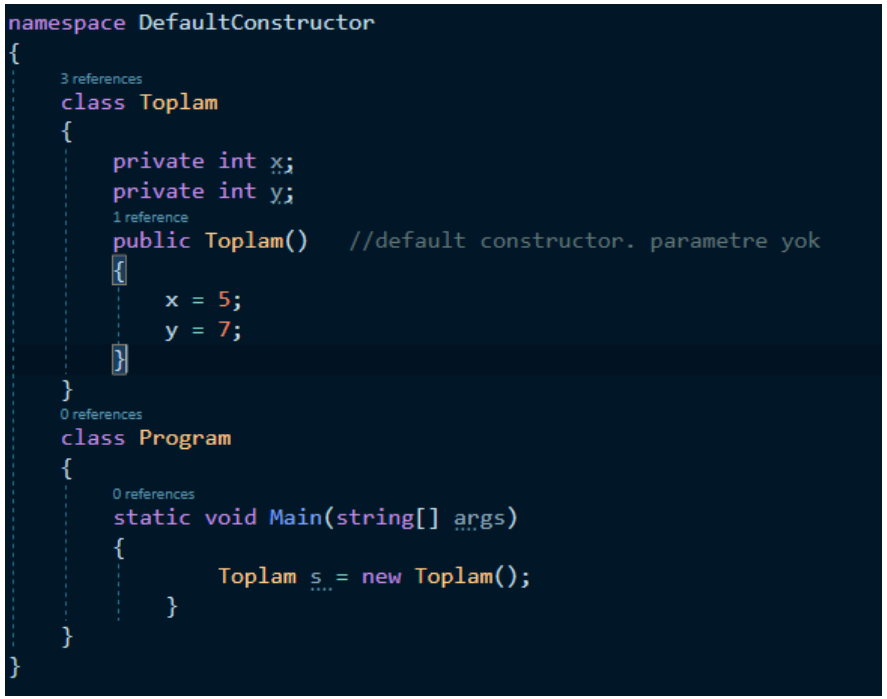
```
public personel() //Sınıf ismiyle aynı olduğuna dikkat
{
    Console.WriteLine("Personel nesnesi oluşturuldu");
}
```

Constructor Özellikleri

- Kurucu metot **abstract, const,static** ve **synchronized** olamaz
- Yalnız bir tane static kurucu metot olabilir.
- Dönüş tipine sahip değildir. **void** alamaz.
- Static kurucu parametre alamaz.
- Bir sınıfın birden fazla kurucu metodu olabilir. Constructor method,method overload destekler.

Constructor Çeşitleri

1. **Default Constructor:**Parametresiz constructor. Sınıfın bütün öğelerinin aynı değerlere başlatılmasını sağlar. Yani tüm fieldlar varsayılan değerlerini alarak nesneleri oluşturur. Tüm sayısal tipler (byte, short, int, long, float,double 0), string ve object null değerini alırlar.



```
namespace DefaultConstructor
{
    3 references
    class Toplam
    {
        private int x;
        private int y;
        1 reference
        public Toplam()   //default constructor. parametre yok
        {
            x = 5;
            y = 7;
        }
    }
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
                Toplam s = new Toplam();
            }
    }
}
```

Constructor Çeşitleri

2. **Parameterized Constructor:**Parametreli bir kurucu bir veya daha fazla parametre alabilir. Bu nedenle, farklı sınıf nesnelerini farklı değerlere başlatabilir. Bu, varsayılan kurucuya göre bir avantajdır.Parametreli bir kurucuyu gösteren bir program aşağıdaki gibidir:

```
3 references
class Toplam
{
    private int x;
    private int y;
    1 reference
    public Toplam(int a, int b) |  //parametreli constructor
    {
        x = a;
        y = b;
    }
}
0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        Toplam s = new Toplam(14,10);
    }
}
```

Constructor Çeşitleri

3. **Copy Constructor:** Bir kopya oluşturucu, verileri başka bir nesneden kopyalayarak nesne değerlerini başlatır. Temel olarak eski nesnenin bir kopyası olan yeni bir nesne oluşturur.Bir kopya oluşturucuyu gösteren bir program aşağıda verilmiştir:

```
class Toplam
{
    private int x;
    private int y;

    1 reference
    public Toplam(Toplam s) //copy constructor
    {
        x = s.x;
        y = s.y;
    }
```

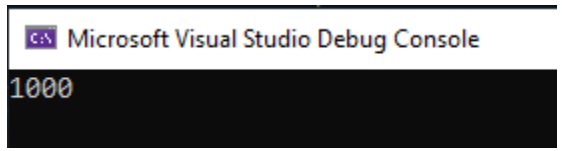
```
class Program
{
    0 references
    static void Main(string[] args)
    {
        Toplam toplam = new Toplam();
        Toplam toplam2 = new Toplam(toplam);
    }
}
```

Constructor Çeşitleri

4. **Private Constructor:** Kurucu metot (constructor), **private** erişim belirleyicisine sahiptir. Diğer sınıfların bu sınıftan kalıtım alması mümkün değildir. Ayrıca bu sınıftan nesne oluşturulamaz. Yalnızca static üyeler varsa kullanılır. Singleton Pattern modelinin uygulanmasında kullanılır.

```
public class Singleton
{
    public static int sayac;
    1 reference
    public static int SayacArtir()
    {
        return ++sayac;
    }

}
0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        Singleton.sayac = 999;
        Singleton.SayacArtir();
        Console.WriteLine(Singleton.sayac);
    }
}
```



Constructor Çeşitleri

5. **Base Constructor:** Türetilmiş bir sınıfa bir kurucu metot eklememize yarar. **:base** anahtar sözcüğü ile kullanılır. Böylece türetilmiş sınıf , temel sınıfın kurucu metodunu çağırabilir.

```
class Person
{
    public Person(string name)
    {
        Debug.WriteLine("My name is " + name);
    }
}

class Employee : Person
{
    public Employee(string name, string job)
        : base(name)
    {
        Debug.WriteLine("I " + job + " for money.");
    }

    public Employee() : this("Jeff", "write code")
    {
        Debug.WriteLine("I like cake.");
    }
}
```

```
var foo = new Person("ANaimi");
// output:
//  My name is ANaimi

var bar = new Employee("ANaimi", "cook food");
// output:
//  My name is ANaimi
//  I cook food for money.

var baz = new Employee();
// output:
//  My name is Jeff
//  I write code for money.
//  I like cake.
```

Constructor Çeşitleri

6. **This Constructor:** Bir kurucu metot çağrısının aynı sınfın başka kurucu metodunu çağırması için **:this()** kullanılır. **:this** ifadesi, sınıfın kurucu metotları arasında kodun paylaşılmasına izin verir. This anahtar sözcüğü, derleyiciye ilk kurucunun üstündeki kurucuya bir çağrı eklemesidir.

```
class Programci
{
    int yasi;
    string adi;
    string soyadi;
    string kullandigiDil;
    // Hic parametre almayan bir yapılandırıcı..
    1 reference
    public Programci() : this(40, "Adem", "AKKUŞ", "C#")
    {
    }
    1 reference
    public Programci(int yasi, string adi, string soyadi, string kullandigiDil)
    {
        Console.WriteLine($"Adı: {adi} Soyadı: {soyadi} Yaşı:{yasi} Programlama Dili: {kullandigiDil}:");
    }
}
```

```
Microsoft Visual Studio Debug Console
Adı: Adem Soyadı: AKKUŞ Yaşı:40 Programlama Dili: C#:
```

Constructor Çeşitleri

7. **Static Constructor:** static constructor direk çağrılamaz. Ancak normal bir kurucu çağırdığımızda, statik kurucu otomatik olarak çağrılır.

```
using System;

namespace Constructor {

  class Car {

   // static constructor
   static Car () {
    Console.WriteLine("Static Constructor");
   }

    // parameterless constructor
    Car() {
     Console.WriteLine("Default Constructor");
   }

    static void Main(string[] args) {

      // call parameterless constructor
      Car car1 = new Car();

      // call parameterless constructor again
      Car car2 = new Car();

      Console.ReadLine();
    }
  }
}
```

```
Car car1 = new Car();
```

```
Static Constructor
Default Constructor
Default Constructor
```

Constructor Çeşitleri

7. **Static Constructor:**Statik kurucu, programın yürütülmesi sırasında yalnızca bir kez çağrılır. Bu yüzden yapıcıyı tekrar çağırdığımızda sadece normal yapıcı çağrılır.

```
using System;

namespace Constructor {

  class Car {

   // static constructor
   static Car () {
    Console.WriteLine("Static Constructor");
   }

    // parameterless constructor
    Car() {
     Console.WriteLine("Default Constructor");
   }

    static void Main(string[] args) {

      // call parameterless constructor
      Car car1 = new Car();

      // call parameterless constructor again
      Car car2 = new Car();

      Console.ReadLine();
    }
```

```
Car car1 = new Car();
```

```
Static Constructor
Default Constructor
Default Constructor
```

> **Not** : Bir sınıfta sadece bir tane statik kurucumuz olabilir. Herhangi bir parametreye veya erişim değiştiricisine sahip olamaz.

Property (Özellik)

Property: sınıf değişkeninin (field) bir uzantısıdır ve sınıf değişkeninin değerini, uygulamalarımıza harici erişim yolunu etkilemeden okumak, yazmak veya değiştirmek için bir mekanizma sağlar.

C# 'da, özellikler (properties) erişimciler belirleyiciler(access modifiers) adı verilen bir veya iki kod bloğu içerebilir ve bunlara erişim belirleyiciler ve ayar erişimcisi adı verilir. get ve set erişimcilerini kullanarak, sınıf değişkenlerinin dahili uygulamasını değiştirebilir ve gereksinimlerimize bağlı olarak ona dış erişim yolunu etkilemeden bunu açığa çıkarabiliriz.

Tipi	Tanımı
Yazma-Okuma	Hem **get** hem de **set** erişimcileri içeren bir özellik, daha sonra onu okuma-yazma özelliği olarak adlandıracağız.
Sadece Yazma	Sadece **get** erişimcisi içeren bir özellik, daha sonra onu salt okunur özellik olarak adlandıracağız.
Sadece Okuma	Sadece **set** erişimci içeren bir özellik, o zaman onu salt yazılır özellik olarak adlandıracağız.

Property (Özellik)

Property: sınıf değişkeninin (field) bir uzantısıdır ve sınıf değişkeninin değerini, uygulamalarımıza harici erişim yolunu etkilemeden okumak, yazmak veya değiştirmek için bir mekanizma sağlar.

C# ‘da, özellikler (properties) erişimciler belirleyiciler(access modifiers) adı verilen bir veya iki kod bloğu içerebilir ve bunlara erişim belirleyiciler ve ayar erişimcisi adı verilir. get ve set erişimcilerini kullanarak, sınıf değişkenlerinin dahili uygulamasını değiştirebilir ve gereksinimlerimize bağlı olarak ona dış erişim yolunu etkilemeden bunu açığa çıkarabiliriz.

```
<erisim_degistirici> <donus_degeri> <ozellik_adi>
{
    get
    {
        // return değeri
    }
    set
    {
        // set değeri
    }
}
```

Property (Özellik)

Aşağıdaki örnekte, “Ad” adında bir özellik tanımladık ve bir özellik değeri döndürmek ve erişimcileri yeni bir değer ayarlamak için ayarlamak için bir alma erişimcisi kullandık. Burada, set erişimcisindeki value anahtar sözcüğü, set erişimcisi tarafından atanan bir değeri tanımlamak için kullanılır.

```
class Kisi
{
  private string ad;
  public string Ad
  {
    get { return ad; }
    set { ad = value; }
  }
}
```

C# ‘da get erişimcisinin yalnızca alan değerini döndürmek veya onu hesaplamak ve döndürmek için kullanılması gerekir, ancak bunu bir nesnenin durumunu değiştirmek için kullanmamalıyız..

Property (Özellik)

Aşağıdaki örnekte, özel değişken adının davranışını **get** ve **set** erişimcileri ile **Ad** adlı bir özelliği kullanarak, set erişimcisini kullanarak **Ad** değerinin yalnızca “Ahmet” e eşit olduğundan emin olmak ve özellik metnini şuna dönüştürmek gibi bazı doğrulamalar gerçekleştirerek genişletiyoruz.Burada “**ad**” alanı özel olarak işaretlenmiştir, bu nedenle bu alanda herhangi bir değişiklik yapmak isterseniz, bunu yalnızca özelliği (Ad) arayarak yapabiliriz.

```
class Kisi
{
  private string ad="Ahmet Can";
  public string Ad
  {
    get { return ad.ToUpper(); }
    set {
          if(value=="Ahmet")
            ad = value;
        }
  }
}
```

Property (Özellik)

Burada “ad” alanı özel olarak işaretlenmiştir, bu nedenle bu alanda herhangi bir değişiklik yapmak isterseniz, bunu yalnızca özelliği (Ad) arayarak yapabiliriz.

C# özelliklerinde, bir özelliğin değerini okurken get erişimcisi çağrılır ve özelliğe yeni bir değer atadığımızda, yeni değeri sağlayan bir bağımsız değişken kullanılarak set erişimcisi çağrılır.Aşağıda, C# programlama dilinde özelliklerin get ve set erişimcilerini çağırma örneği verilmiştir.

```
using System;

class Kisi
{
  private string ad="Ahmet Can";
  public string Ad
  {
    get { return ad.ToUpper(); }
    set {
          if(value=="Ahmet")
            ad = value;
        }
  }
}

class MainClass {
  public static void Main (string[] args) {

    Kisi kisi=new Kisi();
    kisi.Ad="Kemal";
    Console.WriteLine(kisi.Ad);
  }
}
```

Property (Özellik)

C# Sadece Okunabilir Özellik Oluşturma

Daha önce de anlatıldığı gibi, bir özellik tek alma erişimcisini içeriyorsa, onu salt okunur özellik olarak adlandıracağız. Aşağıda, c# programlama dilinde salt okunur özellikler oluşturma örneği verilmiştir.

```
using System;

class Kisi
{
  private int yas=10;
  public int Yas
  {
    get { return yas;  }
  }
}

class MainClass {
  public static void Main (string[] args) {

    Kisi kisi=new Kisi();
    Console.WriteLine(kisi.Yas);
  }
}
```

Property (Özellik)

Get ve set metot olarak düşünebiliriz.

Genel olarak, C# gibi nesne yönelimli programlama dillerinde, alanları özel olarak tanımlamanız ve ardından get ve set erişimcileriyle değerlerine genel bir şekilde erişmek için özellikleri kullanmanız gerekir.

```
class Urun
    {
        private string urunad;
        private string urunkod;
        private double urunfyt;
        //Değer alıp gönderen metodumuz.
        public string UrunAd
        {
            get { return urunad; }
            set { urunad = value; }
        }
```

```
class Person
{
    private string _name;  // the name field
    public string Name    // the Name property
    {
        get => _name;
        set => _name = value;
    }
}
```

Property (Özellik)

Get ve set metot olarak düşünebiliriz.

Genel olarak, C# gibi nesne yönelimli programlama dillerinde, alanları özel olarak tanımlamanız ve ardından get ve set erişimcileriyle değerlerine genel bir şekilde erişmek için özellikleri kullanmanız gerekir.

```
//100-999 arasında değer üretip kullanıcıdan alınan değerle birleştirip
ürün kodu set ediyoruz
public string UrunKod
{
    get { return urunkod; }
    set
    {
        Random rnd = new Random();
        urunkod = value.ToString() + rnd.Next(100, 999).ToString();
    }
}
```

```
//Kullanıcıdan alınan veriyi yuvarlayıp set ediyoruz.
public double UrunFyt
{
    get { return urunfyt; }
    set
    {
        urunfyt = Math.Round(value, 1);
    }
}
```

Property (Özellik)

Get ve set metot olarak düşünebiliriz.

Genel olarak, C# gibi nesne yönelimli programlama dillerinde, alanları özel olarak tanımlamanız ve ardından get ve set erişimcileriyle değerlerine genel bir şekilde erişmek için özellikleri kullanmanız gerekir.

```
//100-999 arasında değer üretip kullanıcıdan alınan değerle birleştirip
ürün kodu set ediyoruz
public string UrunKod
{
    get { return urunkod; }
    set
    {
        Random rnd = new Random();
        urunkod = value.ToString() + rnd.Next(100, 999).ToString();
    }
}
```

```
//Kullanıcıdan alınan veriyi yuvarlayıp set ediyoruz.
public double UrunFyt
{
    get { return urunfyt; }
    set
    {
        urunfyt = Math.Round(value, 1);
    }
}
```