

# BİL 201

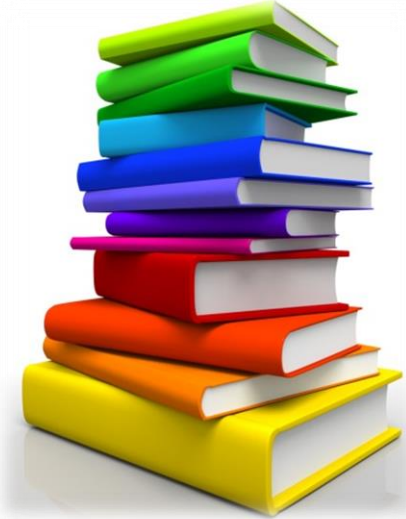
# NESNE YÖNELİMLİ PROGRAMLAMA (NYP)

**DERS #4**

**Öğretim Üyesi: Doç. Dr. Deniz Kılınç**

# BÖLÜM 4 – Nesne Yaratma , Metotlar ve Özellikler

- Bu bölümde aşağıdaki konular anlatılacaktır
  - Sınıflardan Nesne Yaratma,
  - Metotlara Nesne Aktarımı,
  - Properties (Özellikler) Yaratma,
  - Properties Kullanma,
  - Auto-Implemented Properties



# Nesne Yaratma

- Bir sınıfı tanımlamak bir nesne yaratmak anlamına gelmemektedir.
- Sınıf sadece kendisinden yaratılacak bir nesnenin özelliklerini ve davranışlarını belirleyen soyut açıklamasıdır.
- Oluşturmak istediğiniz bir **varlığın** tüm özelliklerini önceden belirlediğinizde, nesne yaratmadan önce bir sınıfın, üyelerini ve metotlarını yaratmalısınız.
- Sınıf tanımlamasını, yeni bir ev inşa etmek için evin **ayrıntılı bir planı** olarak ya da kek yapmak için gerekli olan **kek tarifi** olarak da düşünebilirsiniz.

*Sınıflar oluşturulacak olan herhangi bir nesnenin **ayrıntılı bir planıdır**.*

# Nesne Yaratma (devam...)

- Bir nesneyi yaratmak **iki aşamadan** oluşur.
- **İlk olarak** bir **değişken tanımlar** gibi bir **tip** ve bir tanımlayıcı atamalısınız.
- **Sonrasında** nesneyi yaratabilirsiniz.
- Örneğin; **int** veri tipinde **x** değişkenini **int x**; şeklinde tanımlayabildiğimiz gibi **Personel** nesnesini de, **Personel per**; şeklinde tanımlayabiliriz.
- **Personel** sınıfından **per** adında bir nesne oluşturduğumuzda, derleyiciye **per** adını kullanacağımız **bildirilir**.
- Fakat henüz **per** nesnesinin **barındırdığı değerler için** bellekte bir yer tutulmamıştır.
- Gereken bellek yerini «**rezerve/allocate**» etmek nesneyi **new** operatörünü kullanarak **yaratmamız gereklidir**.

# Nesne Yaratma (devam...)

- **per** adındaki **Personel** nesnesini iki aşamada yaratırız. Öncelikle nesne için bir referans tanımlamalı, sonrasında new operatörü ile derleyicinin **per** nesnesi için yeterli alanı tahsis etmesini sağlarız.

```
Personel per;  
per = new Personel();
```

- Aynı satırda iki işlemi de tanımlayabiliriz:

```
Personel per = new Personel();
```

ifadesindeki **Personel()** bir **metot** gibi gözükmektedir.

- Aslında bu **Personel** nesnesinin **constructor**'ı başka bir deyişle kurucu metodudur.
- Sınıfınız için bir constructor oluşturmazsanız, C# sizin için adı sınıfın adı ile aynı olan bir tane kurucu metod yaratır.

# Nesne Yaratma (devam...)

- Nesnelerin **tanımlayıcıları** (Örn: per nesnesi), nesnelerin bellekteki adreslerinin yerlerini gösterirler.
- Herhangi bir sınıf **referans tipi** (ing.: reference type) olarak **kullanılır**.
- Diğer bir deyişle **bellekteki özel bir adresi** gösteren bir **tip** denebilir.
  - **int**, **double** ve **float** gibi **veri tipleri**, değişkelerin değerlerini tutar;
  - Bu *veri tiplerinin aksine* **referans tipleri** bellekteki adresleri tutarlar.

***Not: C programlamadaki **pointer (işaretçi)** kullanımını hatırlayalım.***

# Metotlara Parametre olarak Nesne Aktarımı

- Bir metoda nesne aktardığımızda aslında o metoda bir referans aktarmış oluruz (pass-by-reference).

*Böylece nesne parametresinde yapılan herhangi bir değişiklik, çağırıldığı metottaki nesneyi de etkiler.*

- Gösterim biçimi aşağıdaki gibidir:

```
void BilgiGoster(Personel p);
```

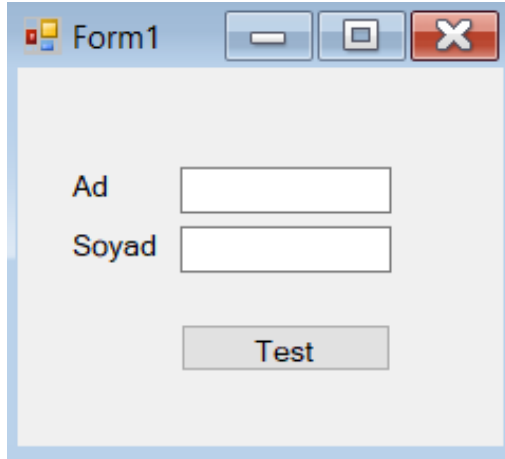
# Örnek1: Metoda Nesne Aktarımı

- Form üzerinde **Personel** sınıfından, **personel** adında bir nesne oluşturunuz.
- Bu nesneyi form üzerindeki PersonelBilgiDegistir metoduna aktarınız.
- Metot içerisinde **personel bilgilerini değiştirip (BÜYÜK harfe dönüştürüp)**, metot dışındaki nesneler üzerinde, *personel bilgisinin değişip değişmediğini* kontrol ediniz.

| Personel                      |
|-------------------------------|
| +Ad: string<br>+Soyad: string |
|                               |



# Örnek1: Metoda Nesne Aktarımı (devam...)



```
public class Personel
{
    public string Ad;
    public string Soyad;
}
```

```
private void PersonelBilgiDegistir(Personel p)
{
    p.Ad = p.Ad.ToUpper();
    p.Soyad = p.Soyad.ToUpper();
}
1 reference
private void BtnTest_Click(object sender, EventArgs e)
{
    Personel personel = new Personel();
    personel.Ad = txtAd.Text;
    personel.Soyad = txtSoyad.Text;

    //metoda parametre olarak nesne yolladık
    PersonelBilgiDegistir(personel);

    //nesne metot dışına çıktığında da değeri değişti
    txtAd.Text = personel.Ad;
    txtSoyad.Text = personel.Soyad;
}
```

# Properties (Özellikler)

# Properties Yaratmak

- **Property**, sınıfın bir **üyesine (değişkenine)** **erişim sağlayan** bir sınıf üyesidir. Property'ler, **üyelerin** nasıl ayarlanacağını ve erişimin nasıl olacağını tanımlar.
- Propertyler, *private üyeler ile public metotlar arasındaki en iyi özellikleri bir arada barındırır.*
- **Public** metotlar gibi private verileri dışarıdan müdahalelere karşı korur.
- Property yarattığınızda, **kullandığınız sözdizimi** daha kolay anlaşılabilir ve **doğal** olur.

# Properties Yaratmak (devam...)

- C# programcıları property'leri **smart fields** olarak adlandırırlar.
- Propertyler, sınıfın üyesine erişildiğinde işleme alınacak olan ifadeleri barındıran **erişimcilere** sahiptir.
- Özellikle propertyler nesnenin üyesinin değerini **ayarlamaya yarayan set erişimcilerine** ve saklanan verilerin **değerlerine erişimini sağlayan get erişimcilerini** içerirler.
- Bir property, set erişimcisine sahipse **yazılabilir**, get erişimcisine sahipse **okunabilir** olmaktadır. Bir property **sadece** get erişimcisine sahipse bu property **sadece okunabilir (read-only)** olur.
- C#'ta get ve set erişimcilerini genellikle **getter** ve **setter** olarak adlandırılırlar.

## Örnek2: Isci Sınıfı

- **kimlikNo** üye değişkenine erişimi sağlayan **KimlikNo** **property**sini tanımlayınız.
- Tanımladığınız **KimlikNo** özelliğini kullanan KarsilamaMesaji() metodunu tanımlayınız.

| Isci                             |
|----------------------------------|
| -kimlikNo: int<br>+KimlikNo: int |
| +KarsilamaMesaji(): string       |

# Örnek2: Isci Sınıfı

```
public class Isci
{
    private int kimlikNo;
    0 references
    public int KimlikNo
    {
        get
        {
            return kimlikNo;
        }
        set
        {
            kimlikNo = value;
        }
    }
    0 references
    public string KarsilamaMesaji()
    {
        return "Hoş geldiniz " + kimlikNo + " numaralı çalışanımız...";
    }
}
```

# Properties Yaratmak (devam...)

- Property tanımlanmasında değişken tanımlanmasına benzer bir şekilde;
  - **Erişim belirleyici**, **veri tipi** ve **tanımlayıcı(ad)** içermektedir.
  - Ayrıca **kıvrık parantezler** içerisinde tanımlanan ifadeleri barındıran bir metot içerir.
- Property ismi kullandığı **üyenin isminin baş harfi büyük hali** olarak tanımlanır.
  - Örn.: **kimlikNo** üye değişkeni için **KimlikNo**
- Property tanımlayıcısından sonra, *küme parantezi içerisinde erişimciler* (**get** ve **set**) tanımlanır.
- Erişimciler (**get** ve **set**) **metot gibi görünebilir** fakat metotlardaki gibi tanımlayıcılarının yanında **parantez '()' bulunmaz**.

# Properties Yaratmak (devam...)

- **set** erişimcisi, **parametre alan** ve bunu **değişkene atayan** bir metot gibi davranır.
- Fakat *set erişimcisi* bir **metot değildir** ve beraberinde bir parantez ile kullanılmaz.
- **get** erişimcisi ise property ile **ilişkilendirilmiş** üyenin değerini *geriye döndürür*.
- Bir propertynin set ve get erişimcilerinden faydalanırken set ve get anahtar kelimelerini ayrıca kullanmamıza gerek yoktur.
- İşlem yapacağımız propertye **eşittir** operatörü '=' ile *değer atadığımızda set bloğu işleme alınmaktadır.*
- Propertynin adını kullanarak eriştığımızde ise **get bloğu işleme alınır.**



# Properties Yaratmak (devam...)

- Örneğin; **usta** adındaki **Isci** sınıfından türetilmiş bir nesne tanımladığımızda, **KimlikNo** propertiesinin değerini aşağıdaki gibi atarız:

```
Isci usta = new Isci();  
usta.KimlikNo = 888191012;
```

- İkinci ifadede, **KimlikNo** propertysinin değeri **888191012** olarak ayarlanmış olur. Eşitliğin sağındaki değer property'nin set erişimcisine **implicit parametre** olarak gönderilmiştir.

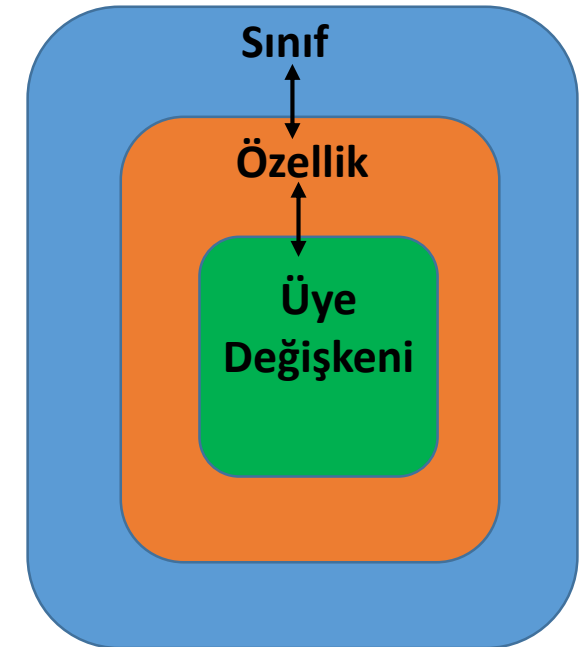
***DEBUG EDEREK GÖRELİM...***

# Properties Yaratmak (devam...)

```
Isci usta = new Isci();  
usta.KimlikNo = 888191012;
```

- 888191012 değeri set bloğuna aktarılmış ve set erişimcisinde **value** adını almıştır.
- Set erişimcisinin içerisinde de value değeri sınıf field'ı olan **kimlikNo** değişkenine atanmıştır.

```
class Isci  
{  
    private int kimlikNo;  
    public int KimlikNo  
    {  
        get  
        {  
            return kimlikNo;  
        }  
        set  
        {  
            kimlikNo = value;  
        }  
    }  
}
```



# Properties Yaratmak (devam...)

- Propertyler `get` erişimcisi sayesinde, basit bir değişken gibi kullanılabilir.
- Örnek: Önceden tanımlanmış olan işçinin kimlik numarası aşağıdaki gibi yazdırılabilir:

```
Isci usta = new Isci();  
//set erişimcisi çalışıyor  
usta.KimlikNo = 888191012;  
  
//get erişimcisi çalışıyor  
MessageBox.Show("Kimlik No: " + usta.KimlikNo);
```

- Fakat usta.**kimlikNo** üye `private` erişim belirleyicisine sahip olduğu için erişilemez ve ekrana yazdırılamaz.

# Properties Yaratmak (devam...)

- Önceki örneklerde **get** ve **set** erişimcileri, üyenin değerini döndürme veya değeri üyeye atama dışında bir işlem yapmamaktadır.
- Eğer kullanılan üye **private** yerine **public** olarak tanımlansaydı **property kullanmamıza gerek kalmazdı.**
- Fakat bu yöntem (Sınıfın üyelerini private, onlara erişmeyi sağlayan metotları public tanımlanması), nesneye yönelik programlamanın **tutarlı olması** için **genel olarak kullanılmaktadır.**

# Properties Yaratmak (devam...)

- Veriyi saklı tutmak ve aynı zamanda
  - verinin değerlerinin **nasıl ayarlandığını** ve
  - kullanıldığını **kontrol edebilmek**,

Nesne Yönelimli Programlamanın çok **önemli bir özelliğidir**.

- Erişimciler, sınıfın bazı üyelerinin **değerlerinin nasıl ayarlanarak döndürüleceğine** veya **üyelere nasıl erişileceğine** dair kısıtlamalar koyarak düzenlenebilir.

# Properties Yaratmak (devam...)

- Örnek: Kimlik numarasının **değer aralığını belirleyen** bir set erişimcisi yandaki gibi yazılabilmektedir.
- Bu kod bloğunda **set** erişimcisi **Isçi** sınıfının **kimlikNo** üyesinin değerinin 1000000'den daha büyük olamayacağını kesinleştiren bir **kısıtlama koyar**.
- Eğer **kimlikNo** değişkenine direkt olarak erişip, değer ataması yapılmasına izin verilseydi, **atanan değer kontrolü yapılamazdı**.
- Tanımlanan sınıf için bir **set** erişimcisi kullandığınızda, izin verilen verilerin değerleri üzerinde tam kontrolü elde edilmiş olunur.

```
get
{
    return kimlikNo;
}
set
{
    if (value < 1000000)
        kimlikNo = value;
    else
        kimlikNo = -1;
}
```

**DEBUG EDEREK GÖRELİM...**

# Auto Implemented Properties

```
public int KimlikNo{get; set;}
```

- Üstteki formatta oluşturulmuş propertylere **auto-implemented property** denir.
- İçerisinde **özelleştirme yapmak istediğiniz** Propertyler, auto-implemented property olarak *yaratılamaz*.
- Ayrıca get ve set erişimcilerinden herhangi birisi kullanılmayacaksa auto-implemented property olarak yaratılamaz.

# Auto Implemented Properties (devam...)

- Sınıfın içerisinde bulunan propertynin kullanacağı üyelere **backing field** denir.
- Auto-implemented property kullandığınızda backing field yaratmaya gerek yoktur.
- Propertynin kullanacağı üye **derleyici tarafından yaratılır**.
- Örnekteki **KimlikNo** propertysi için **kimlikNo** üyesinin yaratılmasına gerek yoktur.
- Derleyici **kimlikNo** üyesini otomatik olarak yaratır ve kullanır.



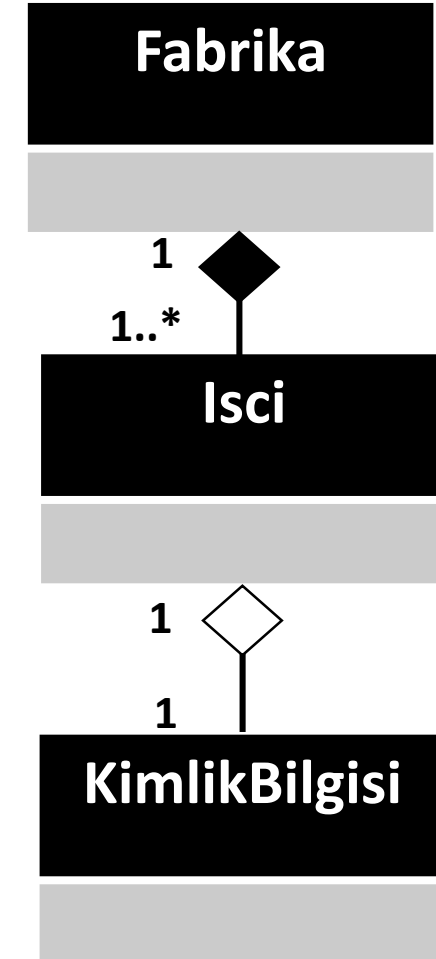
# Örnek3: Isci ve KimlikBilgisi

## Versiyon 1.0 (Senaryo ve Gereksinimler)

- Seyrek fabrikasında çalışan işçilerin hepsi şirket kimlik numarası, maaş ve diğer nüfus kimlik bilgilerine (TCKimlikNo, Ad, Soyad, Doğum Yeri, Doğum Tarihi) sahiptirler. Fabrikaya çalışan ekleme ve çalışan listeleme işlemleri yapılabilir.
- **Özel Not:** Her çalışanın «adını ve soyadını» birleştirerek büyük harfe otomatik çeviren **AdSoyad** özelliği bulunmaktadır.

# Örnek3: Isci ve KimlikBilgisi (devam...)

## Tasarım: UML Sınıf Diyagramı



# Örnek3: Isci ve KimlikBilgisi (devam...)

```
public class KimlikBilgisi
{
    1 reference
    public ulong TCKimlikNo { get; set; }
    2 references
    public string Ad { get; set; }
    2 references
    public string Soyad { get; set; }
    1 reference
    public string AdSoyad
    {
        get
        {
            string adSoyad = (this.Ad + " " + this.Soyad).ToUpper();
            return adSoyad;
        }
    }
    1 reference
    public string DogumYeri { get; set; }
    0 references
    public DateTime DogumTarihi { get; set; }
}
```

```
public class Isci
{
    1 reference
    public int SirketKimlikNo { get; set; }
    1 reference
    public decimal Maas { get; set; }
    2 references
    public KimlikBilgisi Kimlik { get; set; }
}
```

```
public class Fabrika
{
    0 references
    public string Ad { get; set; }
    2 references
    private List<Isci> Calisanlar { get; set; } = new List<Isci>();
    0 references
    public void CalisanEkle(Isci isci)
    {
        Calisanlar.Add(isci);
    }
    0 references
    public List<Isci> CalisanlariListele()
    {
        return Calisanlar;
    }
}
```

## Örnek3: Isci ve KimlikBilgisi (devam...)

```
private void BtnTest_Click(object sender, EventArgs e)
{
    Fabrika fabrika = new Fabrika();
    fabrika.Ad = "Seyrek Fabrikası";

    KimlikBilgisi kimlik = new KimlikBilgisi();
    kimlik.TCKimlikNo = 1221122313;
    kimlik.Ad = "Ahmet";
    kimlik.Soyad = "Demir";
    kimlik.DogumYeri = "İzmir";
    //AdSoyad read-only yani sadece okunabilir
    //kimlik.AdSoyad = "..";

    Isci isci = new Isci();
    isci.Kimlik = kimlik;
    isci.SirketKimlikNo = 9812;
    isci.Maas = 4000;
    MessageBox.Show(isci.Kimlik.AdSoyad);

    fabrika.CalisanEkle(isci);
}
```

# Örnek4: Hali Sınıfının Yaratılması

- **Hali** sınıfında, **Alan** property'si set erişimcisi içermemektedir (read-only).
- **AlanHesapla()** metodu private tanımlanmıştır.
- **Soru**: Nasıl Alan hesaplayacağız?

| Hali                   |
|------------------------|
| +Alan: int {Read-Only} |
| +Genislik: int         |
| +Uzunluk: int          |
| -AlanHesapla()         |

# Örnek4: Hali Sınıfının Yaratılması (devam...)

```
public class Hali
{
    private int alan;
    1 reference
    public int Alan
    {
        get
        {
            AlanHesapla();
            return alan;
        }
    }
    2 references
    public int Genislik { get; set; }
    2 references
    public int Uzunluk { get; set; }
    1 reference
    private void AlanHesapla()
    {
        alan = Uzunluk * Genislik;
    }
}
```

```
private void BtnTest_Click(object sender, EventArgs e)
{
    Hali hali = new Hali();
    hali.Genislik = 4;
    hali.Uzunluk = 3;
    MessageBox.Show("Alan: " + hali.Alan);
}
```

# EK: Auto-Implemented Prop. Nasıl Read-Only Olur?

- Eğer sadece **okunabilir** (**read-only**) auto implemented propertyler yaratmak istiyorsak, set erişimcisini **private** olarak tanımlamalıyız.

## Örnek:

```
public int KimlikNo { get; private set; }
```

- Böylece **KimlikNo** property'sine değer ataması yapılamaz.
- Fakat kendi sınıfındaki metotlar değer ataması yapabilirler.

# Yararlanılan Kaynaklar

- Sefer Algan , HER YÖNÜYLE C# , Pusula Yayıncılık, İstanbul, 2003
- Volkan Aktaş, HER YÖNÜYLE C# 5.0 , Kodlab Yayıncılık, İstanbul, 2013
- Milli Eğitim Bakanlığı "Nesne Tabanlı Programlama", 2012



# İyi Çalışmalar...

**Doç. Dr. Deniz Kılınç**

[deniz.kilinc@bakircay.edu.tr](mailto:deniz.kilinc@bakircay.edu.tr)

[drdenizkilinc@gmail.com](mailto:drdenizkilinc@gmail.com)

[www.denizkilinc.com](http://www.denizkilinc.com)