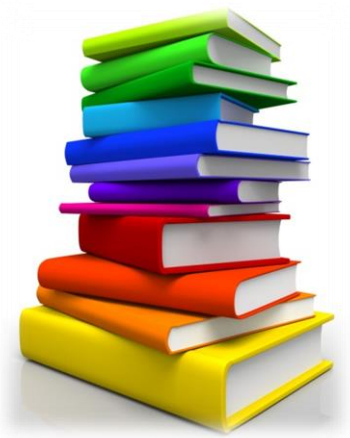
YZM 2105
Nesneye Yönelik Programlama

Yrd. Doç. Dr. Deniz KILINÇ
Celal Bayar Üniversitesi
Hasan Ferdi Turgutlu Teknoloji Fakültesi
Yazılım Mühendisliği

BÖLÜM - 6

Kalıtım (Inheritance) - II



Bu bölümde;

- Temel Sınıfların Metotlarını Ezme
- Çok biçimlilik

ile ilgili konular anlatılacaktır.

Temel Sınıfın Metotlarını Ezme (**Override**) ve Çok Biçimlilik (**Polymorphism**)

Temel Sınıfın Metotlarını Ezme

- Önceden var olan bir sınıftan miras alınarak, genişletilmiş yeni bir sınıf oluşturduğunuzda, yeni sınıf içerisinde temel sınıfın *tüm özellik ve metotları* tanımlanmış olur.
- Bazen parent sınıfın üyeleri, özellikleri ve metotları tam olarak child sınıftan yaratılan nesneler için ***uygun olmayabilir***.
- Farklı işlemler yapan fakat aynı isimdeki özellik veya metotların kullanımına **çok biçimlilik** (polymorphism) denmektedir

Temel Sınıfın Metotlarını Ezme (devam...)

Çok biçimlilik, "birçok form içeren" anlamına gelmektedir.

Aynı isimde *olmalarına rağmen* farklı işlemlerin yer aldığı metotlar için kullanılır.

Temel Sınıfın Metotlarını Ezme (devam...)

- Günlük hayatta çok biçimliliğe **örnek** gösterebilecek çeşitli olaylar bulunmaktadır:
  - Tüm müzik aletleri için "çalmak" eylemi kullanılmasına rağmen, *bir gitarın bir davuldan* farklı bir biçimde çalınıyor olması. (Cal() metotları aynı)
  - Tüm araçlar için "sürmek" eylemi kullanılmasına rağmen, bir *otomobilin* kullanışının bir *bisiklet* kullanımından farklı olması,
  - Tüm okulların "mezun olma koşulları"na sahip olması fakat bir *lise* mezuniyeti ile *ilkokul* mezuniyeti koşulları arasında farklılık olması çok biçimliliğe örnek gösterilebilir.

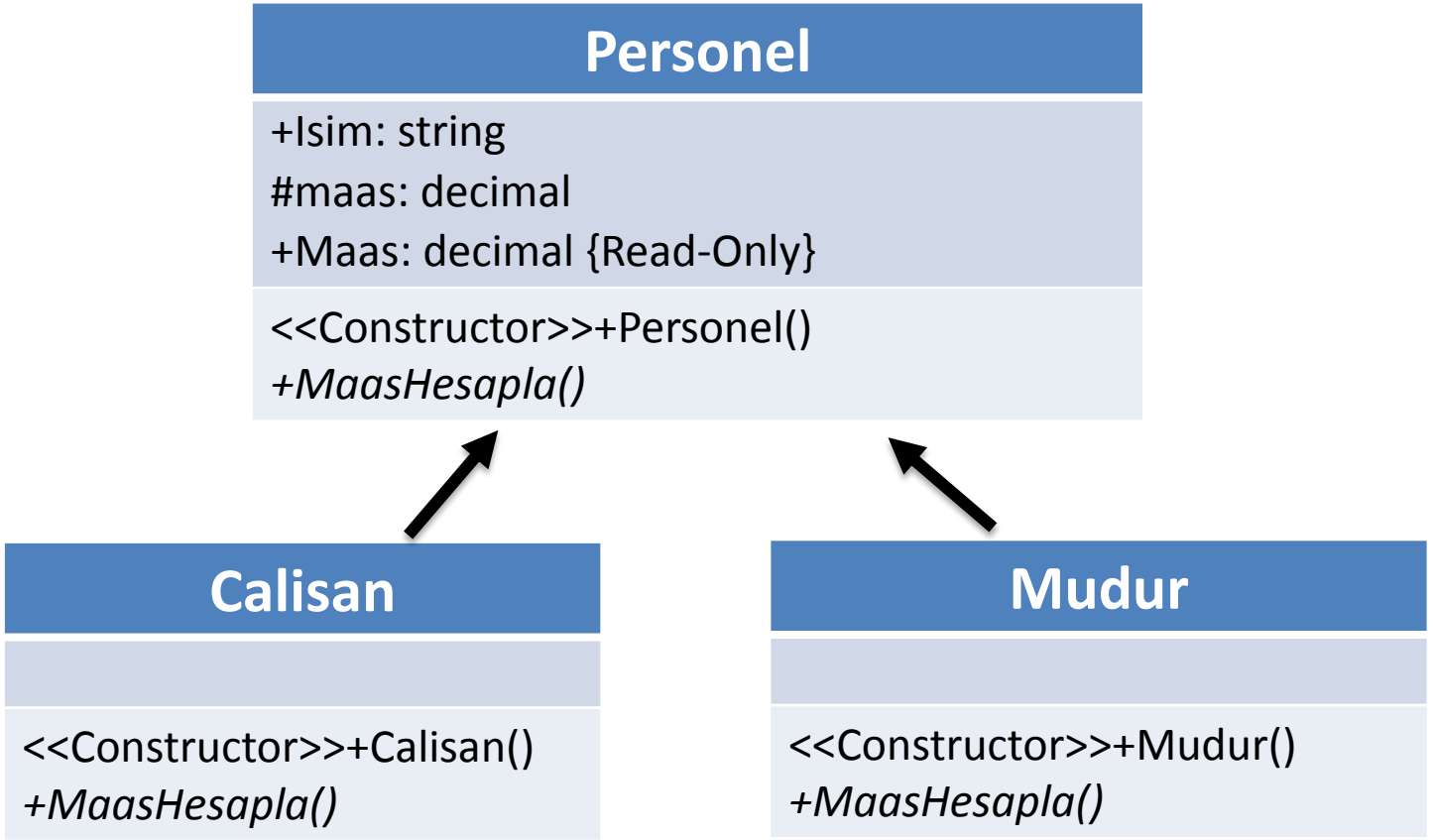
Temel Sınıfın Metotlarını Ezme (devam...)

- Bir **virtual** metot (ya da property) child sınıftaki aynı isme sahip *bir metot tarafından* başına **override** anahtar kelimesi eklenerek ezilmesine olanak sağlar.

```
public class Parent
{
        public virtual void Metot1()
        { …
        }
}

public class Child: Parent
{
        public override void Metot1()
        { …
        }
}
```

Örnek 2: Maaş Hesapla



Örnek 2: Maaş Hesapla

- **Senaryo 1:**
  - Personel maaş hesaplama ile ilgili ***Personel Temel sınıfında*** herhangi özel bir hesaplama yapılmıyor.
  - Çalışan maaşları 1500 TL'dir.
  - Müdür maaşları 4000 TL'dir.
  - Çok biçimlilik kullanarak bu durumu implemente edelim.
  - Form üzerinde nesneleri oluşturarak test ve **DEBUG** işlemi gerçekleştirelim.

Örnek 2: Maaş Hesapla

- **Senaryo 1:**

```
public class Personel
{
    0 references
    public string Isim { get; set; }
    protected decimal maas = 0;
    2 references
    public decimal Maas {
        get {
            return maas;
        }
    }
    1 reference
    public Personel()
    {}
    4 references
    public virtual void MaasHesapla()
    {
        maas = 0;
    }

}
```

```
public class Calisan: Personel
{
    1 reference
    public Calisan()
    {}
    4 references
    public override void MaasHesapla()
    {
        maas = 1500;
    }
}
```

```
public class Mudur: Personel
{
    0 references
    public Mudur()
    {}
    4 references
    public override void MaasHesapla()
    {
        maas = 4000;
    }
}
```

Örnek 2: Maaş Hesapla

- **Senaryo 1:**
  - **Calisan** MaasHesapla() metodu çağırıldığında Temel Sınıf olan **Personel'in** MaasHesapla() metodu çağrılıyor mu?

- **Cevap: Hayır**
- **İhtiyaç olsaydı?**

```
private void Form1_Load(object sender, EventArgs e)
{
    Personel p = new Personel();
    p.MaasHesapla();
    MessageBox.Show(p.Maas.ToString());

    Calisan c = new Calisan();
    c.MaasHesapla();
    MessageBox.Show(c.Maas.ToString());

    Mudur m = new Mudur();
    m.MaasHesapla();
    MessageBox.Show(m.Maas.ToString());
}
```

Örnek 2: Maaş Hesapla

- **Senaryo 2:**
  - Personel maaş hesaplama ile ilgili ***Personel Temel sınıfında*** özel bir hesaplama yapılıyor.
    - **Maaş** = **ASGARIUCRET** (1000)+ **AILEGECIMINDIRIMI (500)**;
  - Çalışan maaşları 1.5 kat.
  - Müdür maaşları 3.5 kat.
  - Çok biçimlilik kullanarak bu durumu implemente edelim.
  - Form üzerinde nesneleri oluşturarak test ve **DEBUG** işlemi gerçekleştirelim.

Örnek 2: Maaş Hesapla

- **Senaryo 2:**

```
public class Personel
{
    private const decimal ASGARIUCRET = 1000;
    private const decimal AILEGECIMINDIRIMI = 500;
    0 references
    public string Isim { get; set; }
    protected decimal maas = 0;
    3 references
    public decimal Maas {
        get {
            return maas;
        }
    }
    1 reference
    public Personel()
    {}
    7 references
    public virtual void MaasHesapla()
    {
        maas = ASGARIUCRET + AILEGECIMINDIRIMI;
    }
}
```

```
public class Mudur: Personel
{
    1 reference
    public Mudur()
    {}
    7 references
    public override void MaasHesapla()
    {
        base.MaasHesapla();
        maas *= 4M;
    }
}
```

```
public class Calisan: Personel
{
    1 reference
    public Calisan()
    {}
    7 references
    public override void MaasHesapla()
    {
        base.MaasHesapla();
        maas *= 1.5M;
    }
}
```

Örnek 3: Öğrenci ve Burslu Öğrenci

Ogrenci
+Isim: string +KrediSayisi: short #dersucreti: decimal +DersUcreti: decimal {Read-Only}
<<Constructor>>+Ogrenci() *+DersUcretiHesapla()*

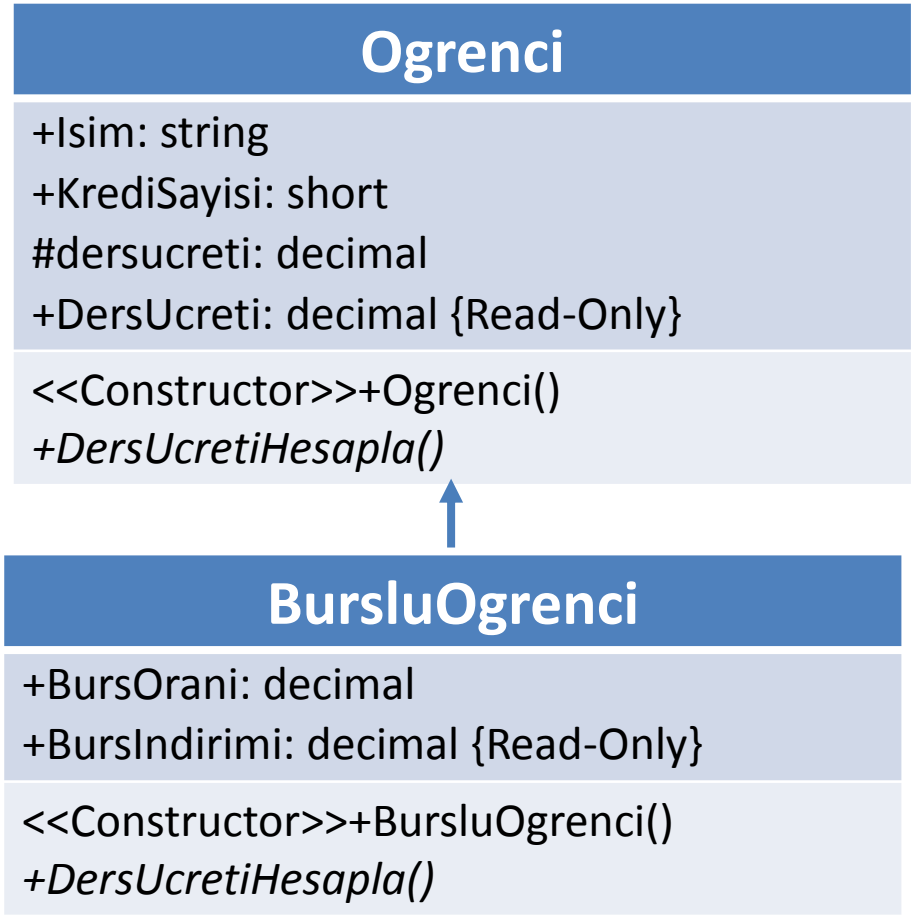
BIRIMDERSUCRETI = 55.75.

- Sınıfı implemente edelim.

Örnek 3: Öğrenci ve Burslu Öğrenci

```
public class Ogrenci
{
    private const decimal BIRIMDERSUCRETI = 55.75M;
    0 references
    public string Isim { get; set; }
    1 reference
    public short KrediSayisi { get; set; }
    2 references
    public decimal DersUcreti {
        get {
            return dersucreti;
        }
    }
    protected decimal dersucreti = 0;
    2 references
    public virtual void DersUcretiHesapla()
    {
        dersucreti = this.KrediSayisi * BIRIMDERSUCRETI;
    }
}
```

Örnek 3: Öğrenci ve Burslu Öğrenci



Örnek 3: Öğrenci ve Burslu Öğrenci

- Ogrenci sınıfından BursluOgrenci sınıfını miras alınarak yaratalım.
- DersUcreti, Ogrenci sınıfı ile *aynı yolla* hesaplanmamaktadır.
- BursluOgrenci sınıfının DersUcreti özelliğinin değeri BursOrani özelliğine göre hesaplanmalıdır.
- Form üzerinde nesneleri oluşturarak test ve **DEBUG** işlemi gerçekleştirelim.

```
BursIndirimi = DersUcreti * BursOrani
```

Örnek 3: Öğrenci ve Burslu Öğrenci

```
public class BursluOgrenci: Ogrenci
{
    2 references
    public decimal BursOrani { get; set; }
    3 references
    public decimal BursIndirimi { get; private set; }
    4 references
    public override void DersUcretiHesapla()
    {
        base.DersUcretiHesapla();
        this.BursIndirimi = DersUcreti * BursOrani;
        dersucreti = DersUcreti - BursIndirimi;
    }
}
```

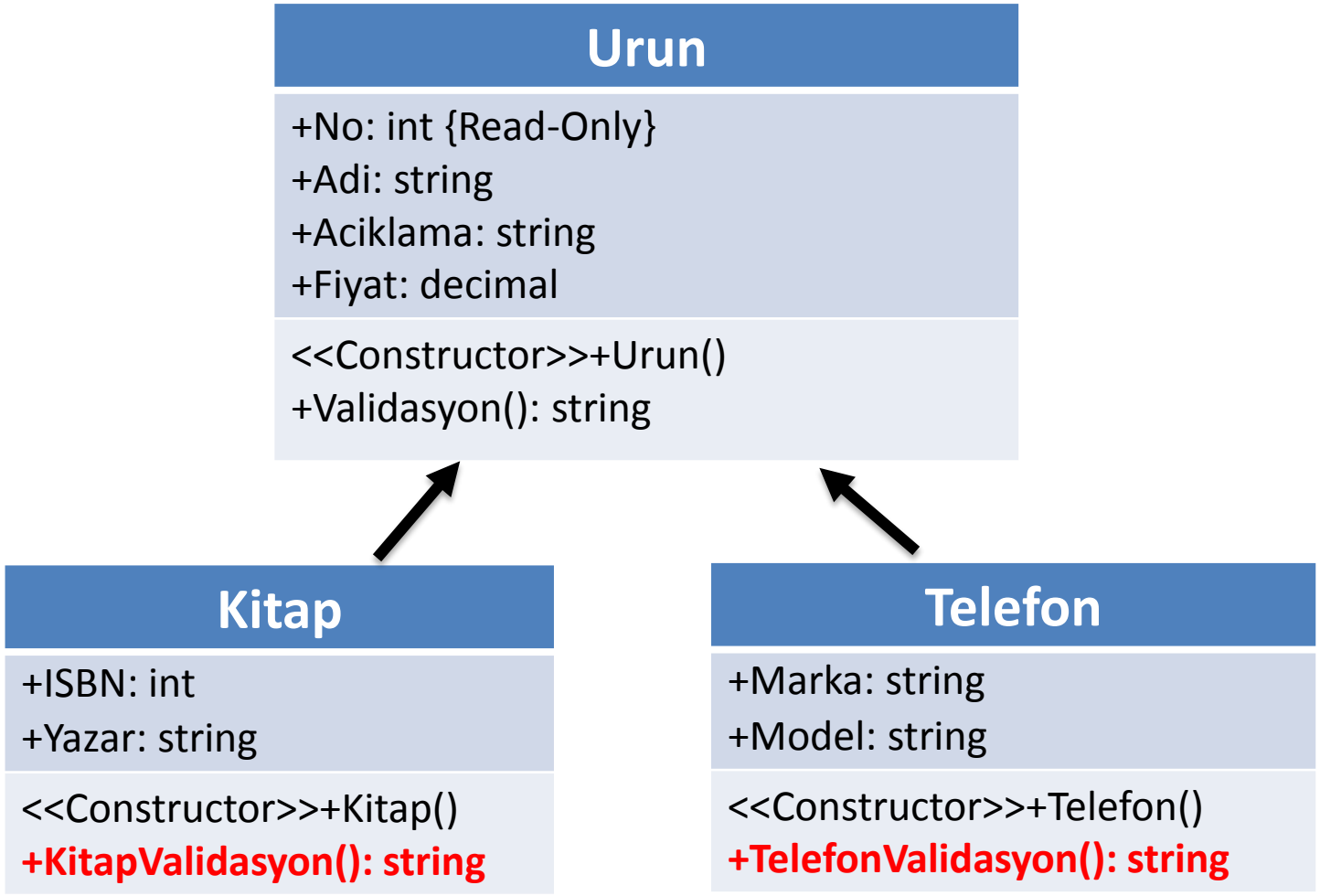
Örnek 3: Öğrenci ve Burslu Öğrenci

```
private void btnTest_Click(object sender, EventArgs e)
{
    Ogrenci o = new Ogrenci();
    o.KrediSayisi = 14;
    o.DersUcretiHesapla();
    MessageBox.Show(o.DersUcreti.ToString());

    BursluOgrenci b = new BursluOgrenci()
    {
        BursOrani = 0.1M,
        KrediSayisi = 14,
    };
    b.DersUcretiHesapla();
    MessageBox.Show("Burs Indirimi:" + b.BursIndirimi.ToString() +
                    "\nDers Ucreti:" + b.DersUcreti.ToString());

}
```

Örnek 4: İlk Kalıtım Örneği - Hatırla !



Örnek 4: İlk Kalıtım Örneği - Hatırla !

```
public string Validasyon()
{
    string hataMesaji = "";
    if ((this.Adi == "") || (this.Adi == null))
        hataMesaji += "Ad özelliği boş olamaz." + Environment.NewLine;
    if (this.Fiyat == 0)
        hataMesaji += "Fiyat özelliği 0 olamaz." + Environment.NewLine;
    return hataMesaji;
}
```

```
public string KitapValidasyon()
{
    string hataMesaji = "";
    if (this.ISBN == 0)
        hataMesaji += "ISBN özelliği boş olamaz." + Environment.NewLine;
    if ((this.Yazar == "") || (this.Yazar == null))
        hataMesaji += "Yazar özelliği boş olamaz." + Environment.NewLine;
    //Üst sınıfın Validasyon metodu çağrılıyor
    return hataMesaji + this.Validasyon();
}
```

Örnek 4: İlk Kalıtım Örneği - Hatırla !

- Eski tasarım *tam olarak* doğru değil.
- KitapValidasyon() ve TelefonValidasyon() **isimli** metotları *oluşturmaya artık gerek yok*.
- Bunun yerine, Temel Sınıftaki Validasyon() metodu virtual yapılarak yavru sınıflarda ezilebilir ve NYP (OOP) çerçevesinde daha **doğru bir tasarım gerçekleştirilir**.

Temel Sınıfın Metotlarını Ezme (devam...)

- Kalıtılarak yaratılmış olan sınıfta sanal bir metodu ezmek zorunlu değildir; Kalıtılarak oluşturulmuş bu sınıf, temel sınıfın versiyonunu da kullanabilir.

- Yavru sınıf, temel sınıfın metotlarına base anahtar sözcüğü ile erişebilir.

Yararlanılan Kaynaklar

- Sefer Algan, HER YÖNÜYLE C# , Pusula Yayıncılık, İstanbul, 2003
- Milli Eğitim Bakanlığı, «Nesne Tabanlı Programlama», 2012
- Joyce Farrel, An Introduction to Object - Oriented Programming, Cengage Learning, 2011
- http://www.AlgoritmaveProgramlama.com

İYİ ÇALIŞMALAR…

Yrd. Doç. Dr. Deniz KILINÇ

deniz.kilinc@cbu.edu.tr