BİL 201
NESNE YÖNELİMLİ PROGRAMLAMA (NYP)

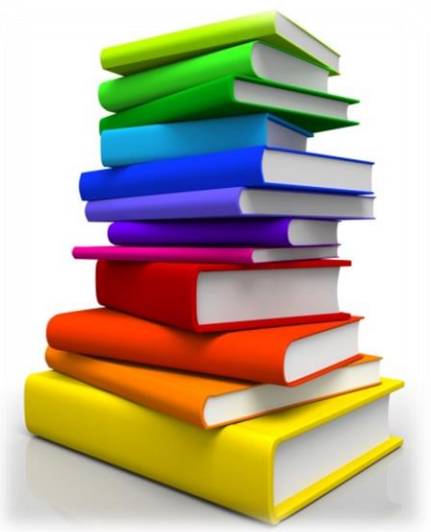
DERS # 10

Öğretim Üyesi: Doç. Dr. Deniz Kılınç

BAKIRÇAY ÜNİVERSİTESİ

BÖLÜM 10 – Arayüzler(Interface)

- Bu bölümde aşağıdaki konular anlatılacaktır.
  - Arayüz tanımı
  - Örnekler
  - Arayüz ve soyut sınıf birlikte kullanımı



Arayüzler

Arayüzler (devam...)

- Bazı Nesne Yönelimli Programlama dilleri, bir yavru (child) sınıfın birden fazla ana sınıfın özelliklerini ve davranışlarını barındırmasına izin vermektedir.
- Birden fazla sınıfın özelliklerini ve davranışlarını miras alma işlemine çoklu kalıtım (multiple inheritence) denmektedir.
- Çoklu kalıtım **zor bir kavramdır** ve kullanıldığında ***çok fazla hata*** ile karşılaşılmaktadır.
- En açık örneği ise parametre alan kurucu metoda sahip iki ayrı sınıfın özelliklerini barındırıyorsa hangisinin kurucu olarak kullanılacağıdır.
  - **Hangi ana sınıf temel alınmalıdır?**

Arayüzler (devam…)

- Bütün bu sıkıntılardan dolayı çoklu kalıtım C# ta engellenmiştir. Fakat C# çoklu kalıtıma arayüz (interface) adında bir **alternatif sunmaktadır**.
- Soyut sınıflar gibi **arayüzler**, herhangi bir sınıfın üyelerinin ya da metotlarının koleksiyonudur.
- Bu metotlar herhangi bir sınıf tarafından, arayüzün soyut metotlarının tanımı sağlandığı sürece (override edildiği sürece) kullanılabilirler.
- Soyut sınıfların içerisinde bazı metotlar soyut olmayabilir.
- Fakat bir **arayüzün** içerisindeki ***tüm metotlar*** soyut olmalıdır.

Arayüzler (devam...)

- Bir arayüz tanımlanırken **interface** anahtar kelimesi kullanılır.

  Arayüz adlandırmaları, başına 'I' harfi eklenerek sonuna da "-able / -ebilen" eki getirilerek yapılır.

```
interface ICalisabilen
{
        string Calis();
}
```

- Herhangi bir sınıf ICalisabilen arayüzünü implemente ettiğinde, ayrıca geriye string ifadesi döndüren **Calis() metodunu** da içermiş olur.

Örnek 1: ICalisabilen Arayüzü

- *ICalisabilen* adında, string Calis() metoduna sahip arayüzü yaratınız.
- Yarattığınız bu arayüzü Ogretmen, Mudur ve Satisci sınıflarına implemente ediniz.
- Calis() metodu sınıflara göre geriye döneceği değerler şu şekilde düzenlenmelidir:

Sınıf Adı	Calis() Metodunun Geriye Döneceği İfade
Ogretmen	«Öğretirim…»
Mudur	«İdarecilik yaparım…»
Satisci	«Satış yaparım…»

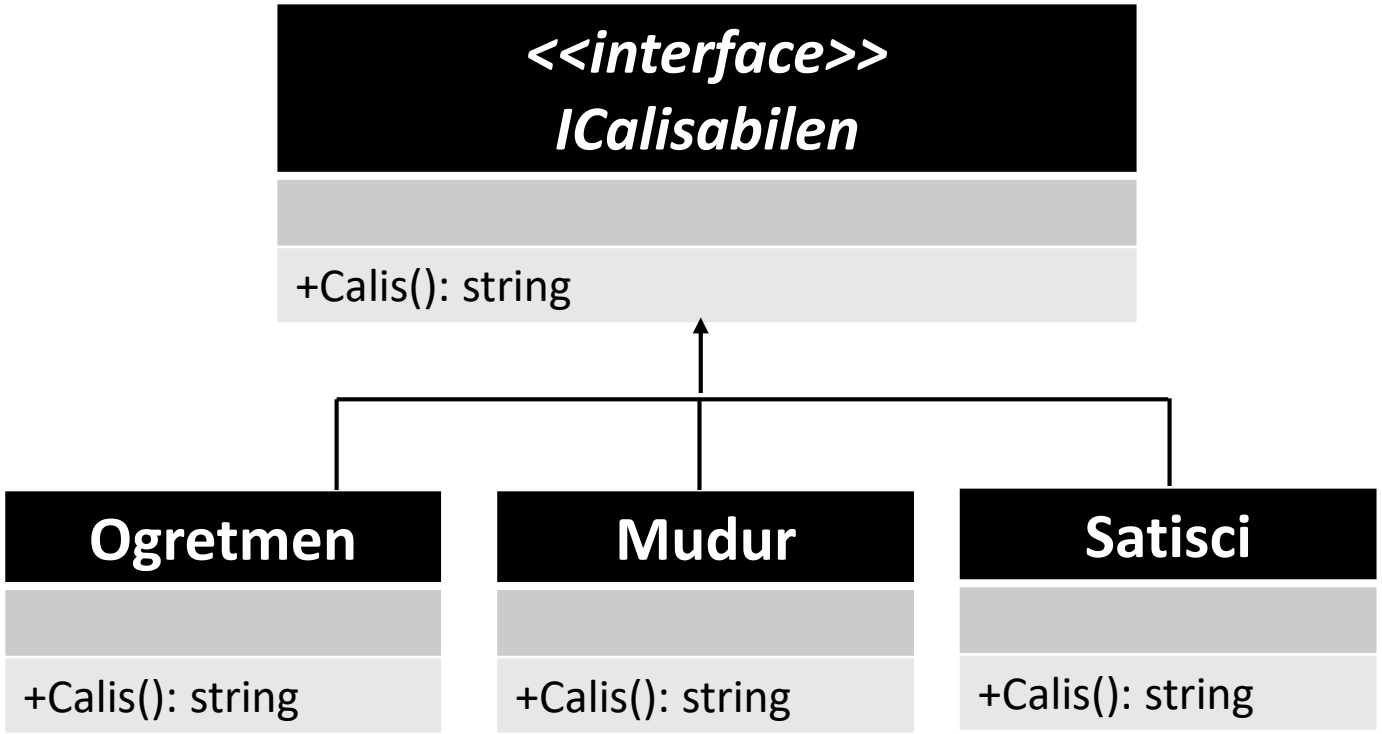
Örnek 1: ICalisabilen Arayüzü

- ICalisabilen arayüzünü implemente edelim (gerçekleştirimini yapalım / kod parçalarını oluşturalım / program satırlarını yazalım):

```
public interface ICalisabilen
{
    0 references
    string Calis();
}
```

Örnek 1: ICalisabilen Arayüzü

- **Senaryo1:** UML diyagramına bakarak implementasyon yapalım, test kodunu yazalım.



Örnek 1: ICalisabilen Arayüzü

```
public class Satisci : ICalisabilen
{
    3 references
    public string Calis()
    {
        return "Satış yaparım...";
    }
}
```

Senaryo1

```
public class Ogretmen : ICalisabilen
{
    3 references
    public string Calis()
    {
        return "Öğretirim...";
    }
}
```

```
public class Mudur : ICalisabilen
{
    3 references
    public string Calis()
    {
        return "İdarecilik yaparım...";
    }
}
```

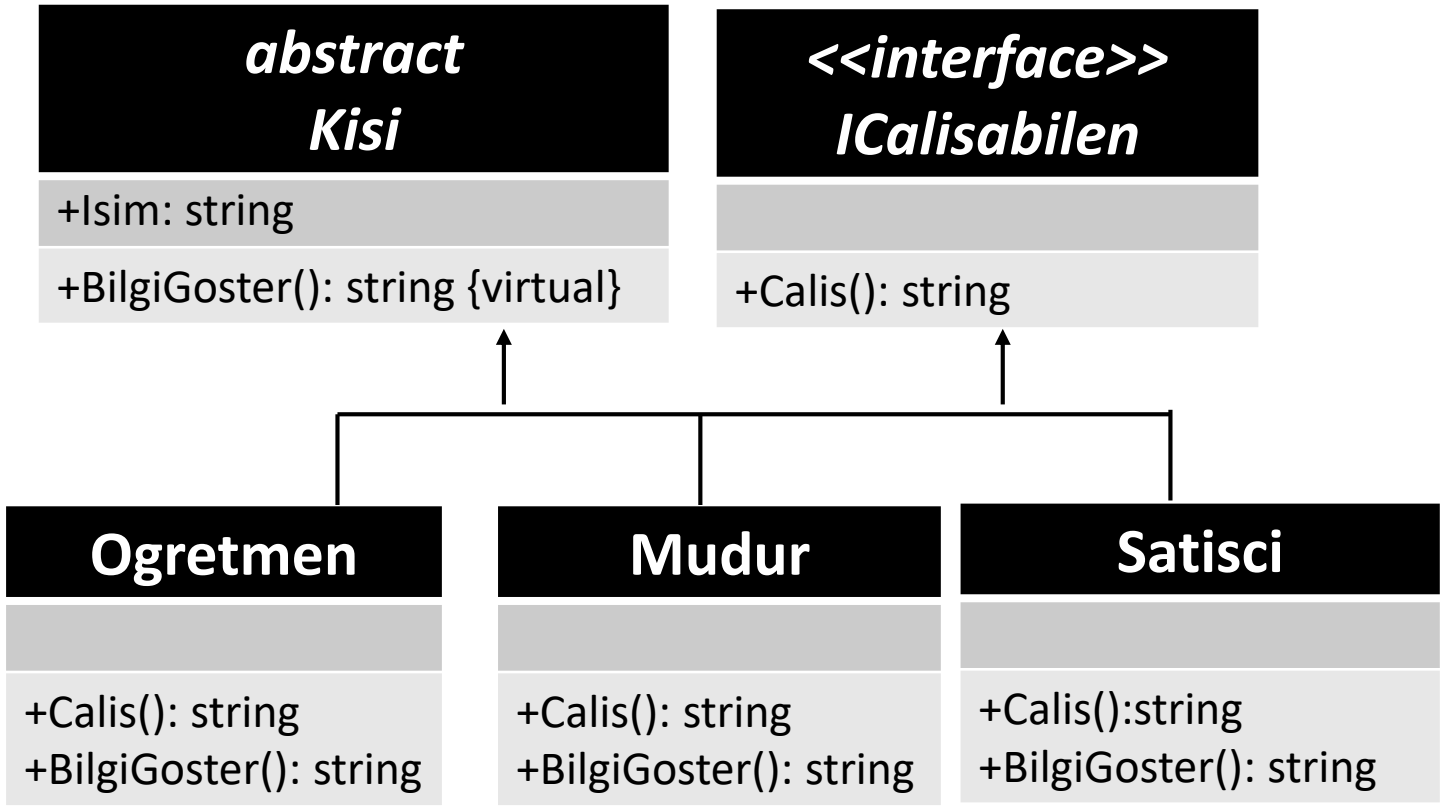
```
private void Form1_Load(object sender, EventArgs e)
{
    Ogretmen ogretmen = new Ogretmen();
    Mudur mudur = new Mudur();
    Satisci satisci = new Satisci();

    MessageBox.Show(ogretmen.Calis());
    MessageBox.Show(mudur.Calis());
    MessageBox.Show(satisci.Calis());
}
```

Arayüzler (devam…)

- ICalisabilen arayüzü her implemente edildiğinde, Calis() metoduna bir **gövde** tanımlanmalıdır.
- Calis() metodu tanımlandığı sınıfa göre işlem yapar.
- Bu işlem **polimorfizm** yani **çok biçimliliktir**.
- Arayüzlerden *nesne yaratılamaz*.
- Soyut sınıfların arayüzlerden farkı soyut olmayan metotlar barındırabilmeleridir.
- Arayüzlerin barındırdığı tüm metotlar **soyut olmalıdır**.

Örnek 1: ICalisabilen Arayüzü

- **Senaryo2:** UML diyagramına bakarak gerçekleştirim yapalım, test kodunu yazalım.



Örnek 1: ICalisabilen Arayüzü

```
public class Ogretmen : Kisi, ICalisabilen
{
    6 references
    public string Calis()
    {
        return "Öğretirim...";
    }
    6 references
    public override string BilgiGoster()
    {
        return base.BilgiGoster();
    }
}
```

```
public class Satisci : Kisi, ICalisabilen
{
    6 references
    public string Calis()
    {
        return "Satış yaparım...";
    }
    6 references
    public override string BilgiGoster()
    {
        return base.BilgiGoster();
    }
}
```

```
public class Mudur : Kisi, ICalisabilen
{
    6 references
    public string Calis()
    {
        return "İdarecilik yaparım...";
    }
    6 references
    public override string BilgiGoster()
    {
        return base.BilgiGoster();
    }
}
```

Senaryo2

```
public abstract class Kisi
{
    1 reference
    public string Isim { get; set; }
    6 references
    public virtual string BilgiGoster()
    {
        return "İsim: " + Isim;
    }
}
```

Örnek 1: ICalisabilen Arayüzü

Senaryo2

```
private void Form1_Load(object sender, EventArgs e)
{
    Ogretmen ogretmen = new Ogretmen();
    ogretmen.Isim = "Ahmet Demir";
    Mudur mudur = new Mudur();
    mudur.Isim = "Mehmet Çelik";
    Satisci satisci = new Satisci();
    satisci.Isim = "İsmet Gümüş";

    MessageBox.Show(ogretmen.BilgiGoster() + " - " +  ogretmen.Calis());
    MessageBox.Show(mudur.BilgiGoster() + " - " + mudur.Calis());
    MessageBox.Show(satisci.BilgiGoster() + " - " + satisci.Calis());
}
```

Arayüzler (devam...)

Bir sınıf sadece tek sınıftan türetilebilir fakat

birden fazla arayüzü gerçekleştirebilir.

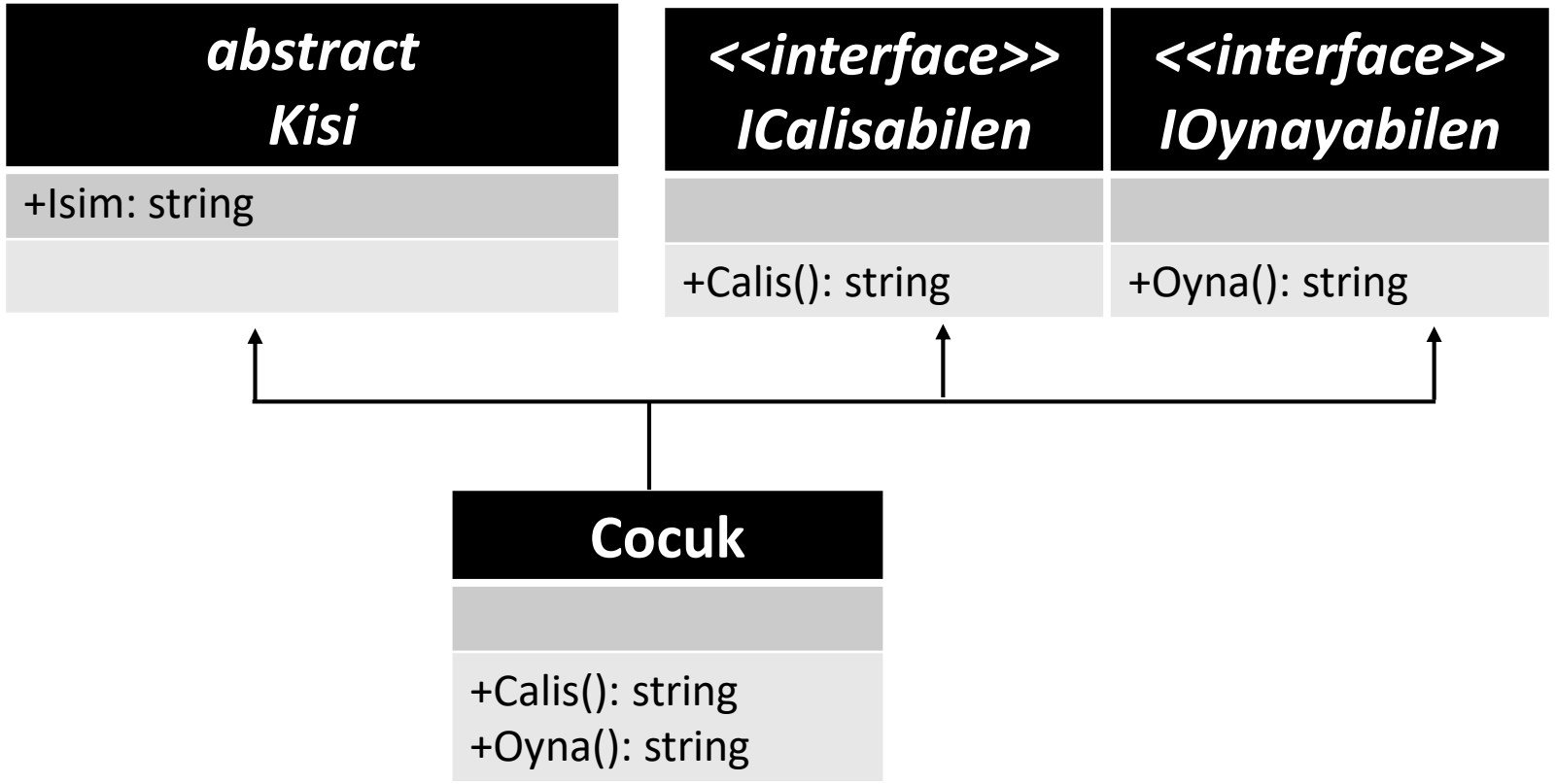
- **Örneğin;** Cocuk sınıfını ele aldığımızda, Kisi sınıfından türetilebilir ve aynı zamanda iki arayüzü de gerçekleştirebilir: ICalisabilen ve IOynayabilen.

```
class Cocuk: Kisi, ICalisabilen, IOynayabilen
```

- Arayüzler bir **sözleşme** olarak düşünülebilir.
- Bir sınıf bu arayüzü implemente ediyorsa **sözleşmeye uymak zorundadır**.

Örnek 2: Çoklu Arayüz Implementasyonu

- UML diyagramına bakarak implementasyon yapalım, test kodunu yazalım.



Örnek 2: Çoklu Arayüz Implementasyonu

```
public abstract class Kisi
{
    0 references
    public string Isim { get; set; }
}
```

```
interface IOynayabilen
{
    1 reference
    string Oyna();
}
```

```
interface ICalisabilen
{
    1 reference
    string Calis();
}
```

```
public class Cocuk : Kisi, ICalisabilen, IOynayabilen
{
    1 reference
    public string Calis()
    {
        return "Ders çalışırım...";
    }
    1 reference
    public string Oyna()
    {
        return "Ders oynarım...";
    }
}
```

Arayüzler (devam…)

Başlangıçta ne zaman arayüz (Interface) ne zaman soyut sınıf (Abstract Class) kullanılacağına **karar vermek** *zordur.* Abstract bir sınıfın bütün metotlarını *abstract yaparak* onu bir interface gibi kullanabiliriz.

- Tipik olarak, child sınıflara *bazı özelliklerin ve metotların* ***aktarılması*** *isteniyor* ve bunlardan bazılarının ezilmesi gerekiyorsa soyut sınıflar kullanılır.
- Her metot ezilmesi zorunluysa, arayüz oluşturulur.

Arayüzler (devam...)

Nesnelerin daha spesifik bir örneğini oluşturmak istiyorsak soyut sınıflar, davranışlarının benzemesini istiyorsak arayüzler yaratılır.

- **Abstract** sınıfların genellikle IS-A (dır, dir) ilişkilerinde, kalıtım(inheritance) özelliğini kullanarak kod tekrarını azaltmak için kullanılır.
- **Interface** sınıfların ise daha çok CAN-DO (yapabilir..) tarzı ilişkilerde değişen kavramları uygulamadan soyutlamak için kullanılır.

Arayüzler (devam...)

```
1 reference
interface IDoJob
{
    1 reference
    void DoJob1();
    1 reference
    void DoJob2();
    1 reference
    void DoJob3();
}
```

```
public class Worker : IDoJob
{
    1 reference
    private void Job1InterProcess()
    {
        //Todo
    }
    1 reference
    public void DoJob1()
    {
        Job1InterProcess();
    }
    1 reference
    public void DoJob2()
    {
        throw new NotImplementedException();
    }
    1 reference
    public void DoJob3()
    {
        throw new NotImplementedException();
    }
}
```

Problem:

- DoJob2 ve DoJob3 fonksiyonlarının gerçekleştirimi yapılmamış.
- Worker sınıfının sadece DoJob1'e ihtiyacı var.

Çözüm:

- SOLID tasarım prensinpleri
- Interface Segregation (ISP)
- Nesneler asla ihtiyacı olmayan metotları içeren interface'leri implemente etmeye zorlanmamalıdır.

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
- Joyce Farrel, An Introduction to Object - Oriented Programming, Cengage Learning, 2011

İyi Çalışmalar...

Doç. Dr. Deniz Kılınç

deniz.kilinc@bakircay.edu.tr

drdenizkilinc@gmail.com

www.denizkilinc.com