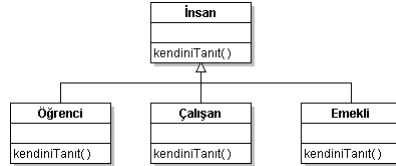


## KALITIM İLE İLGİLİ ÖZEL KONULAR

### ÇOK BİÇİMLİLİK (POLYMORPHISM) ve YENİDEN TANIMLAMA (OVERRIDING)

- İstersek kalıtımla geçen metotların gövdesini değiştirebiliriz dedik.
  - Bu işleme yeniden tanımlama (overriding) adı verilir.
- Üst sınıftan bir nesnenin beklendiği her yerde alt sınıftan bir nesneyi de kullanabiliriz dedik.



- kendiniTanıt() metodunun alt sınıflarda gösterilmesine gerek yoktu ama vurgulamak için yaptım.

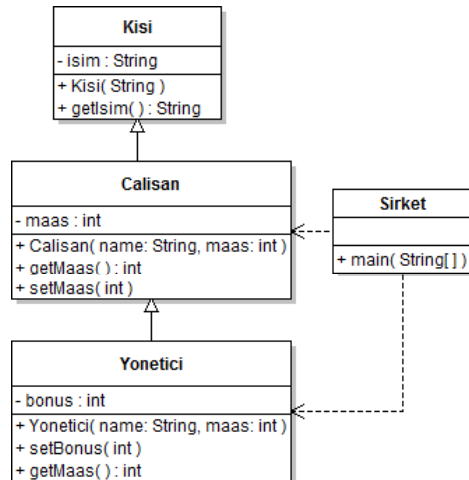
- İnsan türünden bir dizi düşünelim, elemanları İnsan ve alt sınıflarından karışık nesnelere olsun. Dizinin tüm elemanlarına kendini tanıttığımızda ne olacak?
  - Çalışma anında doğru sınıfın metodu seçilir.
  - Bu çalışma biçimine de çok biçimlilik (polymorphism) denir.
- Peki, üst sınıfın altta yeniden tanımladığımız bir metoduna eski yani üst sınıftaki hali ile erişmek istediğimizde ne yapacağız?
  - Bu durumda da super işaretçisi ile üst sınıfa erişebiliriz!

1

## KALITIM İLE İLGİLİ ÖZEL KONULAR

### ÇOK BİÇİMLİLİK (POLYMORPHISM) ve YENİDEN TANIMLAMA (OVERRIDING)

- Örnek kalıtım ağacı: Kişi – Çalışan – Yönetici
  - Ve bunları kullanan sınıf: Şirket
  - UML sınıf şeması:



-

## KALITIM İLE İLGİLİ ÖZEL KONULAR

### ÇOK BİÇİMLİLİK (POLYMORPHISM) ve YENİDEN TANIMLAMA (OVERRIDING)

- Kaynak kodlar:

```
package ders04;
public class Kisi {
    private String isim;
    public Kisi( String name ) { this.isim = name; }
    public String getIsim( ) { return isim; }
}
```

```
package ders04;
public class Calisan extends Kisi {
    private int maas;
    public Calisan( String name, int maas ) {
        super( name );
        this.maas = maas;
    }
    public int getMaas( ) { return maas; }
    public void setMaas( int salary ) { this.maas = salary; }
}
```

3

## KALITIM İLE İLGİLİ ÖZEL KONULAR

### ÇOK BİÇİMLİLİK (POLYMORPHISM) ve YENİDEN TANIMLAMA (OVERRIDING)

- Kaynak kodlar (devam):

```
package ders04;

public class Yonetici extends Calisan {
    private int bonus;

    public Yonetici( String name, int maas ) {
        super( name, maas );
        bonus = 0;
    }

    public void setBonus( int bonus ) {
        this.bonus = bonus;
    }

    public int getMaas( ) {
        return super.getMaas( ) + bonus;
    }
}
```

**DİKKAT: super.super olmaz!!!**

4

## KALITIM İLE İLGİLİ ÖZEL KONULAR

### ÇOK BİÇİMLİLİK (POLYMORPHISM) ve YENİDEN TANIMLAMA (OVERRIDING)

- Kaynak kodlar (devam):

```
package ders04;
public class Sirket {
    public static void main(String[] args) {
        Calisan[] calisanlar = new Calisan[3];
        Yoneticici mudur = new Yoneticici( "Oktay Sinanoğlu", 10000 );
        mudur.setBonus( 2500 );
        calisanlar[0] = mudur;
        calisanlar[1] = new Calisan( "Attila İlhan", 7500 );
        calisanlar[2] = new Calisan( "Ümit Zileli", 6000 );
        for( Calisan calisan : calisanlar )
            System.out.println( calisan.getIsim() + " " +
                calisan.getMaas( ) );
    }
}
```

- For döngüsü dikkatinizi çekti mi?
- Ödev: Sirket sınıfı çok 'yapısal'. Bunu 'nesneye dayalı' hale getirin! Son tasarımınızın UML şemasını da çizin.

5

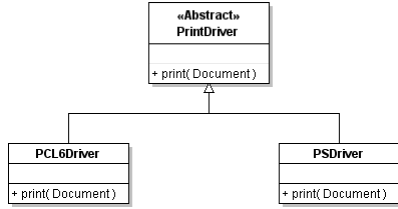
## KALITIM İLE İLGİLİ ÖZEL KONULAR

### SOYUT SINIFLAR

- Soyut sınıflardan nesne oluşturulamaz.
- Ancak soyut sınıfın alt sınıflarından nesnelere kullanılabilir.
- Soyut sınıflar da normal (concrete) sınıflar gibi üye alanlar içerebilir.
- Soyut sınıfın metotları soyut veya normal olabilir:
  - Soyut metotların sadece imzası tanımlanır, gövdesi tanımlanmaz.
  - Normal metotlar alışılmış şekilde imza ve gövdesi ile tanımlanır.
  - Soyut ve normal metotlar bir arada olabilir.
- Ne zaman soyut sınıflara gereksinim duyulur:
  - Bir sınıf hiyerarşisinde yukarı çıkıldıkça sınıflar genelleşir. Sınıf o kadar genelleşmiş ve kelime anlamıyla soyutlaşmıştır ki, nesnelere o açıdan bakmak gerekmez.
  - Bir sınıf grubunda bazı metotların mutlaka olmasını şart koşuyorsanız:
    - Bu metotları bir soyut üst sınıfta tanımlar ve söz konusu sınıfları ile bu soyut sınıf arasında kalıtım ilişkisi kurarsınız.
- Soyut sınıfların adı sağa yatık olarak yazılır ancak gösterimde sorun çıkarsa <<STEREOTYPE>> gösterimi.
  - <<...>>: Bir sembol anlamı dışında kullanılmışsa.

## KALITIM İLE İLGİLİ ÖZEL KONULAR

### SOYUT SINIFLAR

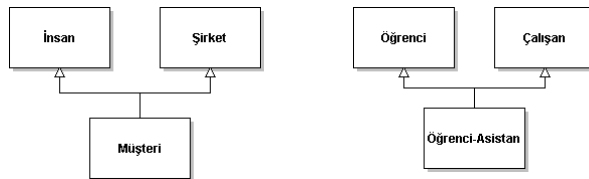


- Üst sınıfta bir yazdırma işleminin olması gerektiği biliniyor ama bu işin nasıl yapılacağı bilinmiyor.
  - Bu nedenle PrintDriver nesnelere bir işimize yaramaz.
  - İşlemin nasıl yapılacağı alt sınıflarda tanımlanmıştır.
  - PCL6 ve PS tipinden birden fazla sürücü bir bilgisayarda kurulu olabilir ve hepsine ortak bir PrintDriver arayüzünden erişilebilir.
- Soyut sınıfların adı sağa yatık olarak yazılır ancak gösterimde sorun çıkarsa <<STEREOTYPE>> gösterimi.
    - <<...>>: Bir sembol anlamı dışında kullanılmışsa.

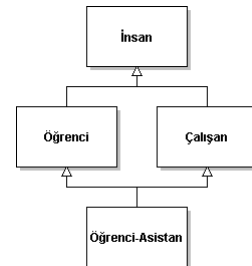
## KALITIM İLE İLGİLİ ÖZEL KONULAR

### ÇOKLU KALITIM

- Bir sınıfın birden fazla üst sınıftan kalıtım yolu ile türetilmesi.
- Alt sınıfın, birden fazla üst sınıfın özelliğini taşıması anlamına gelir.



- Çoklu kalıtım ile ilgili sorunlar:
  - Kalıtım çevrimi (Diamond problem): Orta düzeyde çokbüçümlülük varsa alt düzeyde çokbüçümlü metodun hangi sürümü çalışacak?
  - Her dil desteklemez. Ör: Java!



## NYP İLE İLGİLİ ÖZEL KONULAR

### ADAŞ METOTLAR / ÇOKLU ANLAM YÜKLEME (OVERLOADING)

- Bir sınıfın aynı adlı ancak farklı imzalı metotlara sahip olabileceğini gördük.
- Böyle metotlara adaş metotlar, bu işleme ise çoklu anlam yükleme (overloading) adı verilir.
- Örnek: Çok biçimlilik konusu örneğindeki Yönetici sınıfına bir yapılandırıcı daha ekleyelim:

- Yonetici( String name, int maas, int bonus )

```
public Yonetici( String name, int maas, int bonus ) {  
    super( name, maas );  
    this.bonus = bonus;  
}
```

- Böylece yapılandırıcıya çoklu anlam yüklemiş olduk.
- Bu kez de bu yapılandırıcıyı kullanacak kişi, maaş ile bonus'u birbirine karıştırmamalı.
- DİKKAT: Çoklu anlam yüklemenin kalıtımla bir ilgisi yoktur. Kalıtım olmadan da adaş metotlar oluşturulabilir, ancak kalıtım olmadan çok biçimlilik ve yeniden tanımlama mümkün değildir.

9

## NYP İLE İLGİLİ ÖZEL KONULAR

### ADAŞ METOTLAR / ÇOKLU ANLAM YÜKLEME (OVERLOADING)

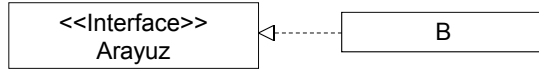
- Soru: Ne zaman böyle bir ekleme işi yapmaya ihtiyaç duyabiliriz?
  - Yanıt: Her yöneticinin mutlaka bonusu olması gerektiği hallerde.
  - Ödev: Şu gereksinimleri gerçekleyecek şekilde Çok biçimlilik konusu örneğindeki Yonetici sınıfını değiştiriniz:
    - Her yöneticinin en az 1000 TL bonusu olmalıdır.
    - Yöneticinin bonusu azaltılamaz, sadece arttırılabilir.
    - Yöneticinin bonusu maaşından hiçbir zaman için fazla olamaz.

10

## NYP İLE İLGİLİ ÖZEL KONULAR

### ARAYÜZLER (INTERFACE)

- Üye alanları olmayan ve tüm metotları soyut olan bir soyut sınıf gibi görülebilir.
- Bir ad altında derlenmiş metotlar topluluğudur.
- Adlandırılmış ancak içeriği tanımlanmamış bir eylemler topluluğu olarak görülebilir.



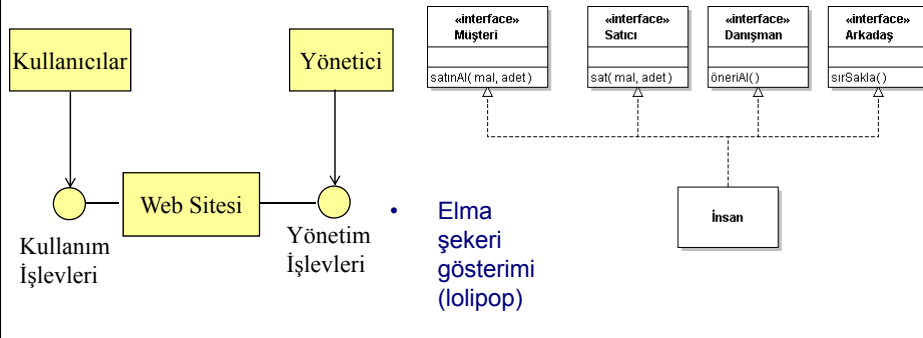
Arayüz Gerçekleme

- Arayüzler ile ilgili kurallar:
  - Arayüzler normal veri içeremez, sadece final üye alanlar içerebilir.
  - Arayüzülerin kurucusu olmaz.
  - Arayüzün tüm metotları public görünürlüğe sahiptir.
  - Bir sınıf birden fazla arayüz gerçekleyebilir.
  - Birden fazla arayüz aynı imzalı metodu içerebilir.
  - Arayüzler, sınıflar tarafından gerçekleştirilir.
  - Bir arayüz birden fazla sınıf tarafından gerçekleştirilebilir.
  - Bir arayüz diğerini genişletebilir.

## NYP İLE İLGİLİ ÖZEL KONULAR

### ARAYÜZLER (INTERFACE)

- Arayüzler neye yararabilir?
  - Nesnenin sorumluluklarını gruplamaya.
  - Nesneye birden fazla bakış açısı kazandırmaya:
    - Farklı tür nesnelere aynı nesneyi sadece kendilerini ilgilendiren açılardan ele alabilir.
    - Farklı tür nesnelere aynı nesneye farklı yetkilerle ulaşabilir.
  - Çoklu kalıtımın yerine kullanılabilir.

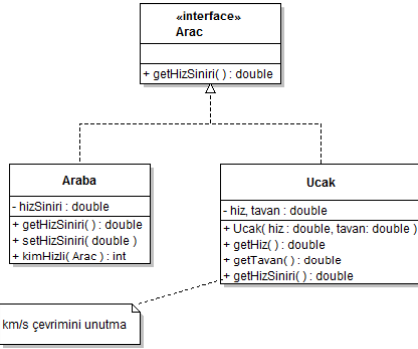


- Elma şekerini gösterimi (lolipop)

## NYP İLE İLGİLİ ÖZEL KONULAR

### ARAYÜZLER (INTERFACE)

- Arayüz örneği: Araç – Araba – Uçak
  - UML sınıf şeması:



- Kaynak kodlar:

```
package ders04.arayuzOrnegi;
public interface Arac {
    public double getHizSiniri( );
}
```

```
package ders04.arayuzOrnegi;
public class Ucak implements Arac {
    private double hiz; //Mach
    private double tavan; //ft.
    public Ucak(double hiz, double tavan){
        this.hiz = hiz;
        this.tavan = tavan;
    }
    public double getHiz( )
    { return hiz; }
    public double getTavan( )
    { return tavan; }
    public double getHizSiniri( )
    { return hiz * 1225; }
}
```

## NYP İLE İLGİLİ ÖZEL KONULAR

### ARAYÜZLER (INTERFACE)

- Kaynak kodlar (devam):

```
package ders04.arayuzOrnegi;
public class Araba implements Arac {
    private double hizSiniri;

    public double getHizSiniri() {
        return hizSiniri;
    }

    public void setHizSiniri(double hizSiniri) {
        this.hizSiniri = hizSiniri;
    }

    public int kimHizli( Arac baska ) {
        if( hizSiniri < baska.getHizSiniri() )
            return -1;
        else if( hizSiniri == baska.getHizSiniri() )
            return 0;
        else
            return 1;
    }
}
```

## NYP İLE İLGİLİ ÖZEL KONULAR

### ARAYÜZLER (INTERFACE)

- Kaynak kodlar (devam):

```
package ders03.iliskiler;
import ders04.arayuzOrnegi.*;

public class AnaProgram05 {

    public static void main(String[] args) {
        Araba mercedes = new Araba();
        mercedes.setHizSiniri(216.5);
        Ucak f16 = new Ucak( 2.1, 60000 );
        int sonuc = mercedes.kimHizli( f16 );
        if( sonuc == -1 )
            System.out.println("F-16 daha hızlı");
        else if( sonuc == 0 )
            System.out.println("İki aracın da hızı aynı");
        else
            System.out.println("Mercedes daha hızlı");
    }
}
```

## NYP İLE İLGİLİ ÖZEL KONULAR

### ARAYÜZLER (INTERFACE)

- Örnek programın sıralama şeması: