

Java'da Soyutlama (“Abstraction”) ve Çok-biçimlilik (“Polymorphism”)

BBS-515 Nesneye Yönelik Programlama

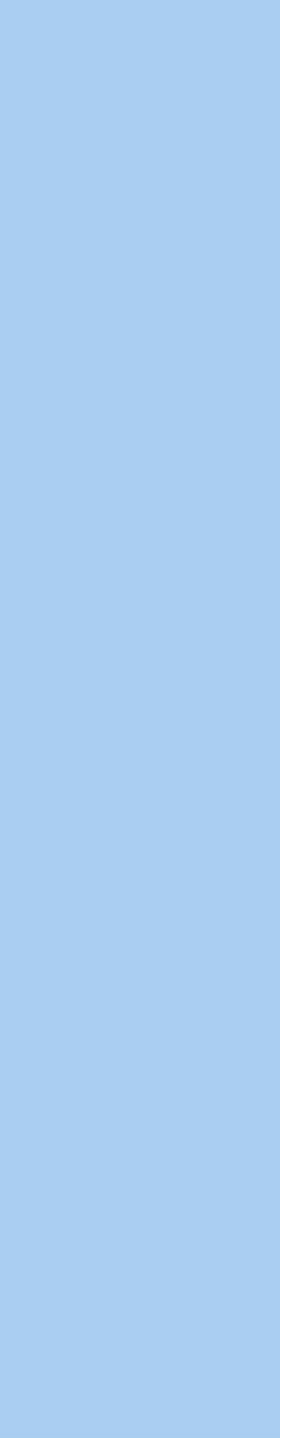
Ders #9 (16 Aralık 2009)

■ Geçen ders:

- ▶ Java Applet'lerde bileşen yerleştirme türleri ("applet layouts")
- ▶ Java'da "Awt" ve "Swing" kütüphane bileşenleri
- ▶ Örnekler

■ Bu ders:

- ▶ Soyutlama ("abstraction")
- ▶ Arayüz ("Interface") tanımlama
- ▶ Çok-biçimlilik ("polymorphism")
- ▶ Örnekler



Soyutlama (“Abstraction”)

Soyut (“Abstract”) Sınıf

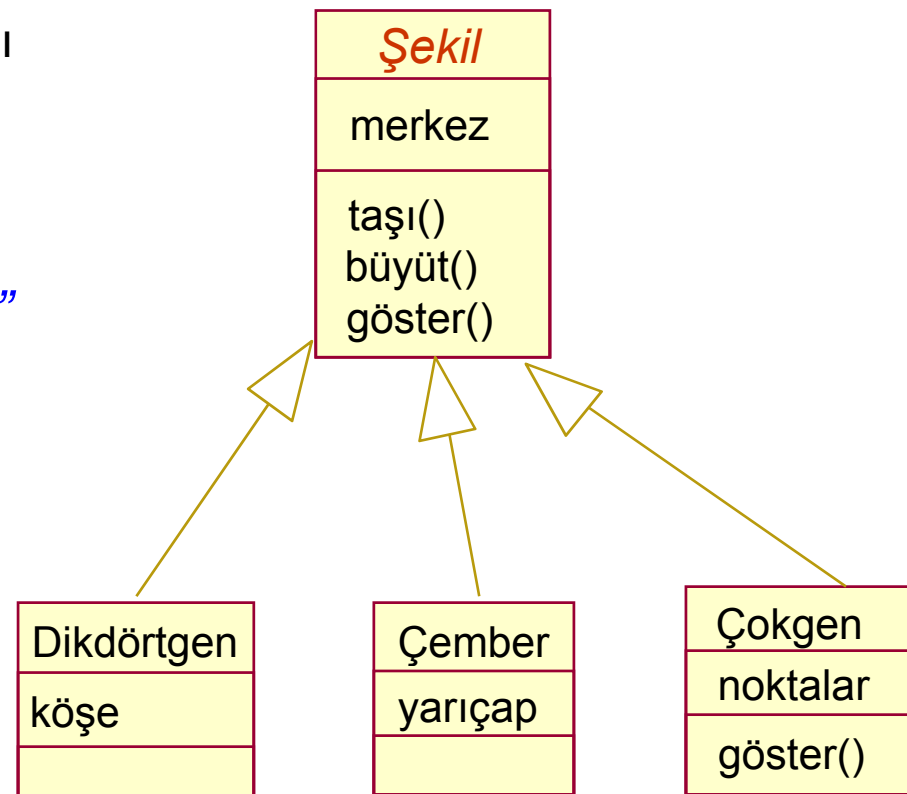
- Alt sınıfların ortak özelliklerini ve işlevlerini taşıyan bir üst sınıf oluşturmak istersek ve gerçek dünyada bu sınıftan bir nesne yoksa, üst sınıfı “soyut sınıf” olarak tanımlarız.
 - ▶ Örnek: “Memeli” sınıfından direkt bir nesne oluşturulmaz; ancak alt sınıfları tanımlanarak onlardan nesneler oluşturulur.
- Soyut sınıfın yöntemlerini, alt-sınıfları tarafından üzerine yazılmak üzere, sadece şablon olarak tanımlayıp içlerini boş bırakabiliriz veya soyut yöntem (“abstract method”) olarak tanımlayabiliriz.
 - ▶ Çağırın sınıflar için arayüz oluşturur.
 - ▶ Alt sınıflar üzerine yazarak işlevlerini tanımlar.

UML'de Soyut (“Abstract”) Sınıf

- Gerçekte nesnesi olmayan bir sınıf, kalıtım ağacında “abstract” (soyut) olarak tanımlanır.

- ▶ Sınıf ismi *yatık* (“*italic*”) yazılır
- ▶ Yeniden kullanma (“reuse”) amaçlı
- ▶ Üzerine yazma için şablon

*“Abstract”
(soyut)
sınıf*



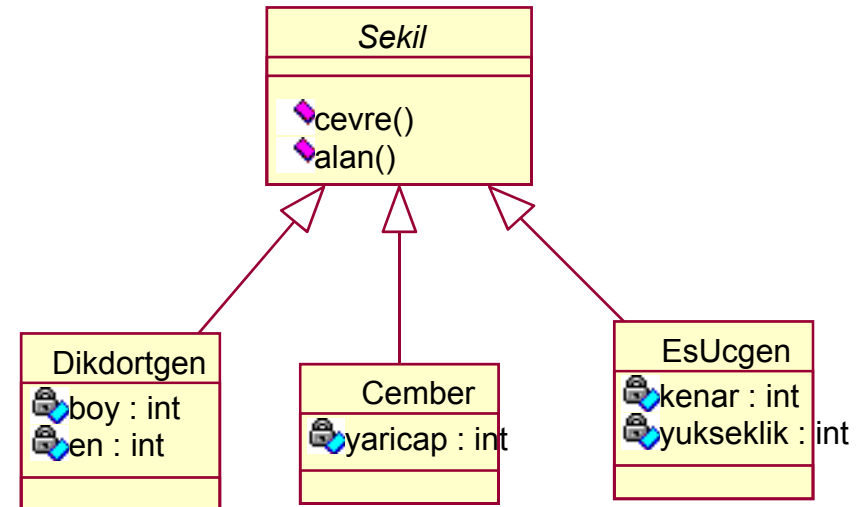
Soyutlama (“Abstraction): Örnek – 1

```
public abstract class Sekil {
```

*Somut
yöntemler
("concrete
methods")*

```
    public int cevre() {  
        // üzerine yazılacak  
        return 0;  
    }
```

```
    public int alan() {  
        // üzerine yazılacak  
        return 0;  
    }  
}
```



```
public abstract class Sekil {
```

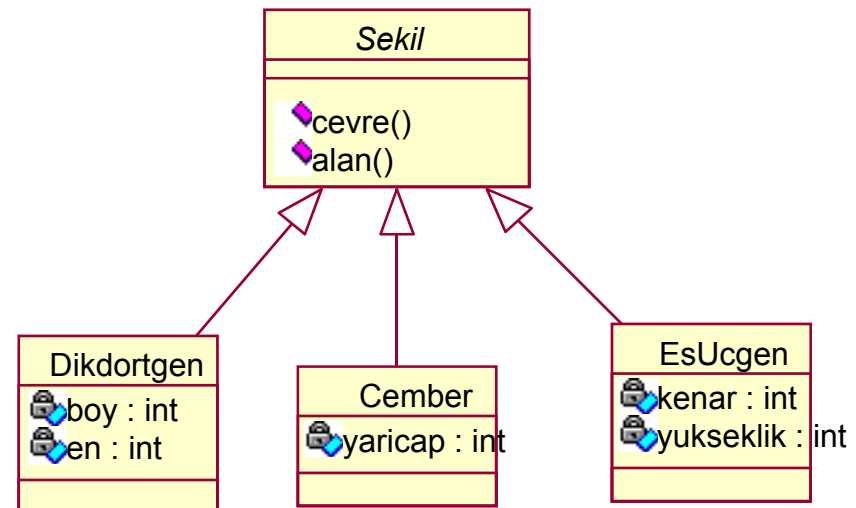
VEYA ;

*Soyut
yöntemler
("abstract
methods")*

```
    public abstract int cevre();  
    public abstract int alan();  
}
```

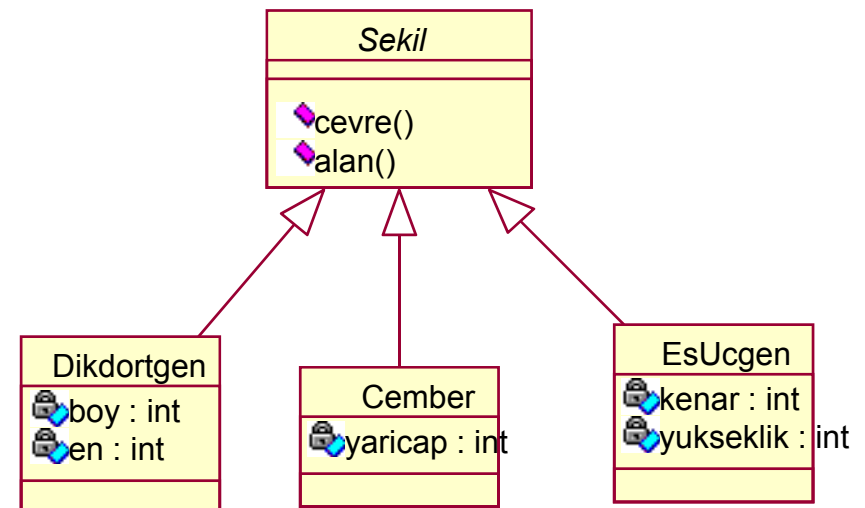
Soyutlama (“Abstraction): Örnek – 1 (devam)

```
public class Dikdortgen extends Sekil {  
    private int boy;  
    private int en;  
    public int cevre() {  
        return (2 * (boy + en));  
    }  
    public int alan() {  
        return (boy * en);  
    }  
}
```



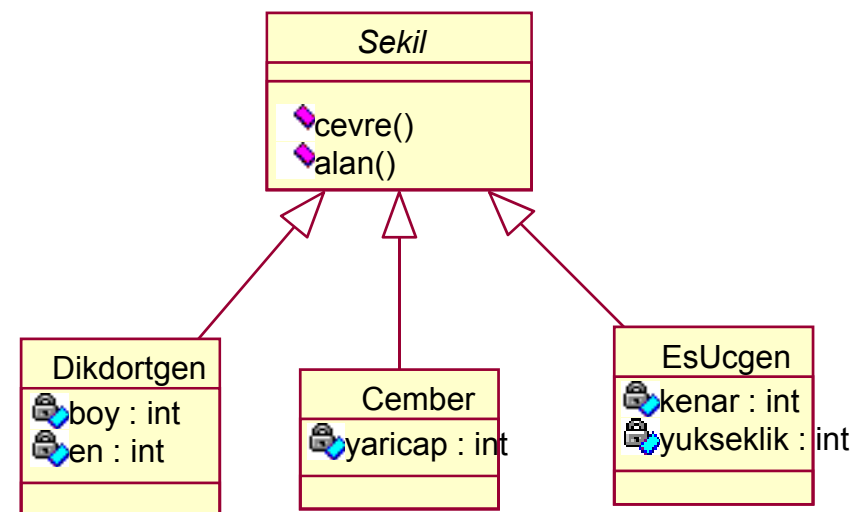
Soyutlama (“Abstraction): Örnek – 1 (devam)

```
public class Cember extends Sekil {  
    private int yaricap;  
    public int cevre() {  
        return (2 * 3 * yaricap);  
    }  
    public int alan() {  
        return ( 3 * yaricap * yaricap);  
    }  
}
```



Soyutlama (“Abstraction): Örnek – 1 (devam)

```
public class EsUcgen extends Sekil {  
    private int kenar;  
    private int yukseklik;  
    public int cevre() {  
        return (kenar * 3);  
    }  
    public int alan() {  
        return ((kenar * yukseklik) / 2);  
    }  
}
```



Soyut Sınıf ve Yöntemlerin Kullanımı

- Soyut sınıflardan nesne oluşturulamaz.
- Soyut yonteme sahip bir sınıfın kendisi de otomatik olarak soyuttur ve öyle tanımlanmak zorundadır.
- Bir soyut sınıfın alt sınıfları, ancak üst sınıfın tanımladığı soyut sınıfların üzerine yazdığı ve onlara birer işlev tanımladığı zaman örneklenebilir (“instantiation”).
 - ▶ Bu durumda alt sınıflar somut sınıf (“concrete class”) olarak adlandırılır.
- Bir soyu sınıf, soyut yöntemlere ek olarak, somut yöntemler de tanımlayabilir.
 - ▶ Bir soyut sınıf sadece somut yöntemleri de içerebilir.
- Eğer bir soyut sınıfın alt sınıfı, o sınıfa ait tüm soyut yöntemleri gerçekleştirmezse; alt sınıf da soyut tanımlanmak zorundadır.
- *static*, *final* ve *private* olarak tanımlı yöntemler, üzerine yazılamadıklarından, soyut olarak tanımlanamazlar.

Soyutlama (“Abstraction): Örnek - 2

```
abstract class A {  
    abstract void beniCagir ();  
    void benideCagir () {  
        System.out.println (“A’nın somut metodu.”);  
    }  
}
```


```
class B extends A {  
    void beniCagir() {  
        System.out.println (“B’nin beniCagir metodu.”);  
    }  
}
```

```
class SoyutDemo {  
    public static void main (String args [] ) {  
        B b = new B ();  
        b.beniCagir ();  
        b.benideCagir (); // Somut (“concrete”) yöntemler de miras alınarak kullanılabilir.  
    }  
}
```

Çıktı:

B’nin beniCagir metodu.

A’nın benideCagir somut metodu.



Arayüz (“Interface”)

Arayüz (“Interface”)

- Java’da çoklu kalıtıma (“multiple inheritance”) izin verilmez. Bunu telafi etmek için **arayüz (“interface”)** kavramı tanımlanmıştır.
 - ▶ Bir sınıfın bir üst-sınıfı olabilir ve birden çok arayüzü gerçekleştirebilir.
- Arayüzler **soyuttur ve doğrudan örneklenemez.**
- Arayüz tanımı sınıfa benzer; ancak **sadece yöntem imzalarını (“method signatures”)** ve **sabit değişkenler (“constant variables”)** içerebilir.
- Bir arayüzü **gerçekleştiren her sınıf, arayüz içinde imzası tanımlanmış yöntemlerin kodlarını yazmak zorundadır (yöntem imzaları aynı olmalıdır.)**
 - ▶ Tüm yöntemlerin kodunu yazmıyorsa sınıf, soyut tanımlanmış olmalıdır.
 - ▶ Tüm yöntemleri yazmıyor ve soyut tanımlanmadı ise derleyici hata verir.

Arayüz (“Interface”): Örnek - 1

```
public class Bisiklet {  
    int vitesSayisi;  
    public Bisiklet (int v) {  
        vitesSayisi = v;  
    }  
    public void print () {  
        System.out.println (“Bu bisikletin “ + vitesSayisi + “ vitesi var.”);  
    }  
}
```

*Ekrana bisiklet
çizdirmek istediğimizi
varsayalım.*

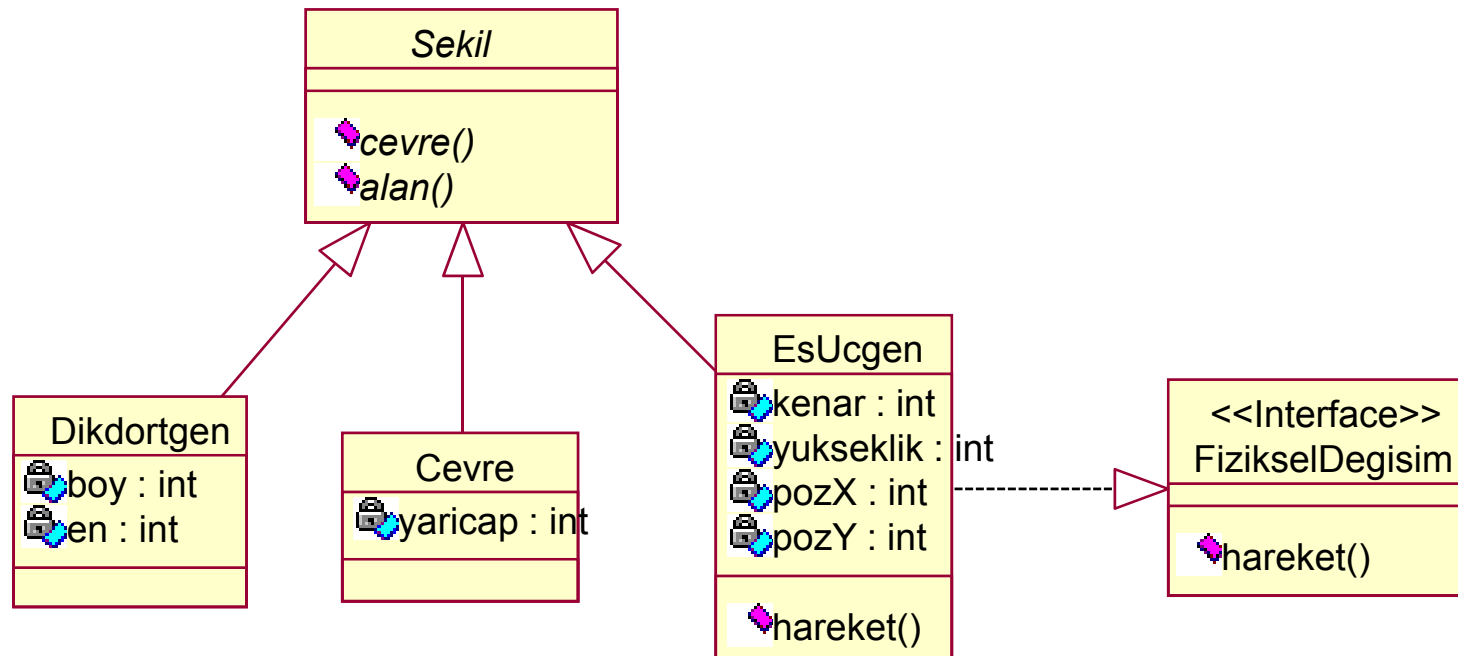
*(Bunun için
java.awt.Canvas sınıfını
kullanmalıyız; ancak çoklu
kalıtım yapamayız.)*

```
public interface Arac {  
    public void print ();  
}
```

```
public class Bisiklet extends Canvas implements Arac {  
    int vitesSayisi;  
    public Bisiklet (int v) {  
        vitesSayisi = v;  
    }  
    public void print () {  
        System.out.println (“Bu bisikletin “ + vitesSayisi + “ vitesi var.”);  
    }  
    public void paint () {  
        // bisikleti çizecek kod  
    }  
}
```

Arayüz (“Interface”): Örnek – 2

- Aşağıdaki “EsUcgen” sınıfından oluşturulan nesnelerin şekil değiştirebileceğini varsayalım. Ancak bu yetenek tüm şekillere değil, sadece “EsUcgen” sınıfına özgü olsun.

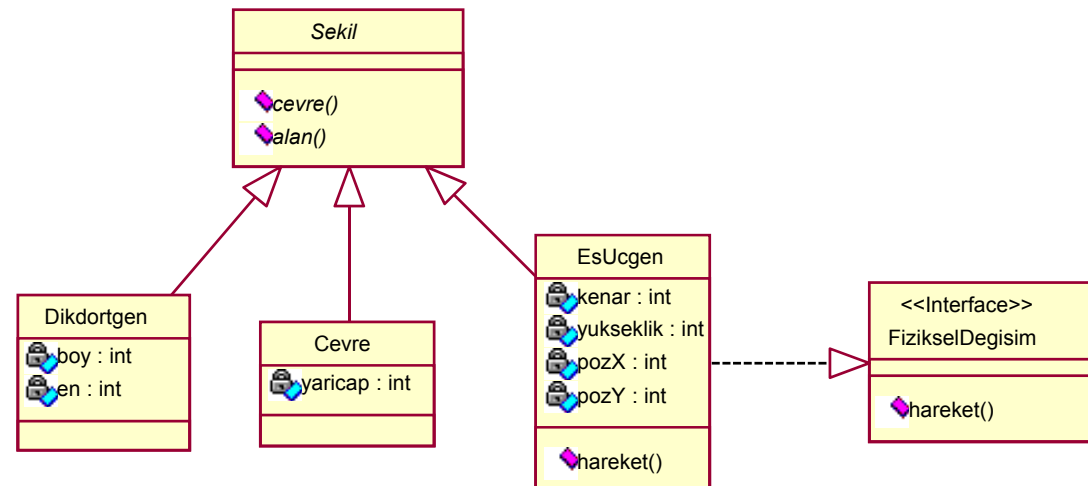


Arayüz (“Interface”): Örnek – 2 (devam)

```
public interface FizikselDegisim {  
    public void hareket();  
}
```

```
public class EsUcgen extends Sekil implements FizikselDegisim {  
    private int kenar;  
    private int yukseklik;  
    private int pozX;  
    private int pozY;
```

```
    public void hareket() {  
        int x,y;  
        System.in.read(x);  
        System.in.read(y);  
        pozX = x;  
        pozY = y;  
    }  
}
```



Arayüz (“Interface”): Örnek - 3

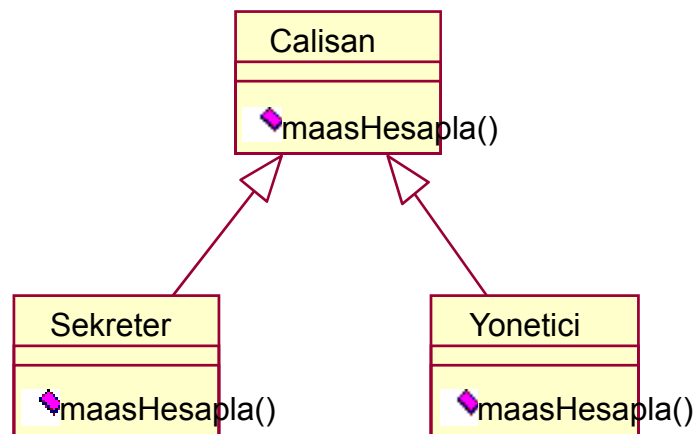
- `java.awt.event.`
 - ▶ `ActionListener.java`



Çok Biçimlilik (“Polymorphism”)

Çok-Biçimlilik

- Bir kalıtım ağacına ait sınıflarda aynı imza (dönüş tipi, ad, parametreler) ile tanımlanmış bir yöntem var ise; Java ortamı çalıştırma zamanında yöntemin hangi sınıfa ait tanımdan çalıştıracağını dinamik olarak belirleyebilir. Bu özelliğe **çok-biçimlilik** (“polymorphism”) denir.



Bu özellik, “if” veya “switch” kullanımına gerek bırakmaz.

Yeni bir işçi alt sınıfı eklendiğinde mevcut kodun değiştirilmesi gerekmez.

Çok-Biçimlilik (“Polymorphism”): Örnek – 1

```
interface Konus {  
    String getAd();  
    String merhaba ();  
}  
abstract class İnsan implements Konus {  
    private final String ad;  
    protected İnsan (String pAd) {  
        this.ad = pAd;  
    }  
    public String getAd() {  
        return this.ad;  
    }  
}
```

Çok-Biçimlilik (“Polymorphism”): Örnek – 1 (devam)

```
class Turk extends İnsan {  
    public Turk (String pAd) {  
        super(pAd);  
    }  
    public String merhaba () {  
        return “Merhaba!”;  
    }  
}  
class İngiliz extends İnsan {  
    public İngiliz (String pAd) {  
        super(pAd);  
    }  
    public String merhaba () {  
        return “Hello!”;  
    }  
}
```

Çok-Biçimlilik (“Polymorphism”): Örnek – 1 (devam)

```
public class Test {  
    public static void main(String[] args) {  
        Insan[] insanlar = { new Turk("Ahmet"),  
                             new Ingiliz ("Marry"),  
                             new Turk ("Ayşe")  
        };  
        for (Insan n : insanlar) {  
            System.out.println(n.getAd() + ": " + n.merhaba());  
        }  
    }  
}
```

Çok-Biçimlilik (“Polymorphism”): Örnek - 2

- Çalışan.java
 - ▶ Maasli.java
 - ▶ Saatli.java
- PolyDemo.java