Kalıtım (“Inheritance”)

BBS-515 Nesneye Yönelik Programlama

Ders #4 (11 Kasım 2009)

İçerik

- Geçen ders:
  - Java'da işleçler ("operators")
  - Java'da kontrol-akış ("control-flow") ve döngü ("loop") deyimleri
  - Java konsol programlarında basit girdi/çıktı ("input/output")

- Bu ders:
  - Kalıtım ("inheritance")

Kalıtım (“Inheritance”)

Kalıtım (“Inheritance”) – 1

- Bazı sınıflar, kendi özeliklerini taşıyan özel tiplere ayrılabilir.
  - Örnek: Bisiklet: dağ bisikleti, yarış bisikleti
  - Dağ bisikleti ve yarış bisikleti; bisiklet sınıfının alt-sınıflarıdır (“sub-classes”).
  - Bisiklet sınıfı; dağ bisikleti ve yarış bisikleti sınıflarının üst-sınıfıdır (“super-class”).

- Her alt-sınıf kendi üst-sınıfının özelliklerini ve işlevlerini taşır (kalıtım - ”inheritance”).
  - Dağ bisikleti ve yarış bisikleti, bisiklet sınıfına ait özellikleri taşır: vites, tekerlek, pedal, vb.
  - Dağ bisikleti ve yarış bisikleti, bisiklet sınıfına ait işlevleri gösterir: hızlanma, fren yapma, vites değiştirme, vb.

Kalıtım (“Inheritance”) – 2

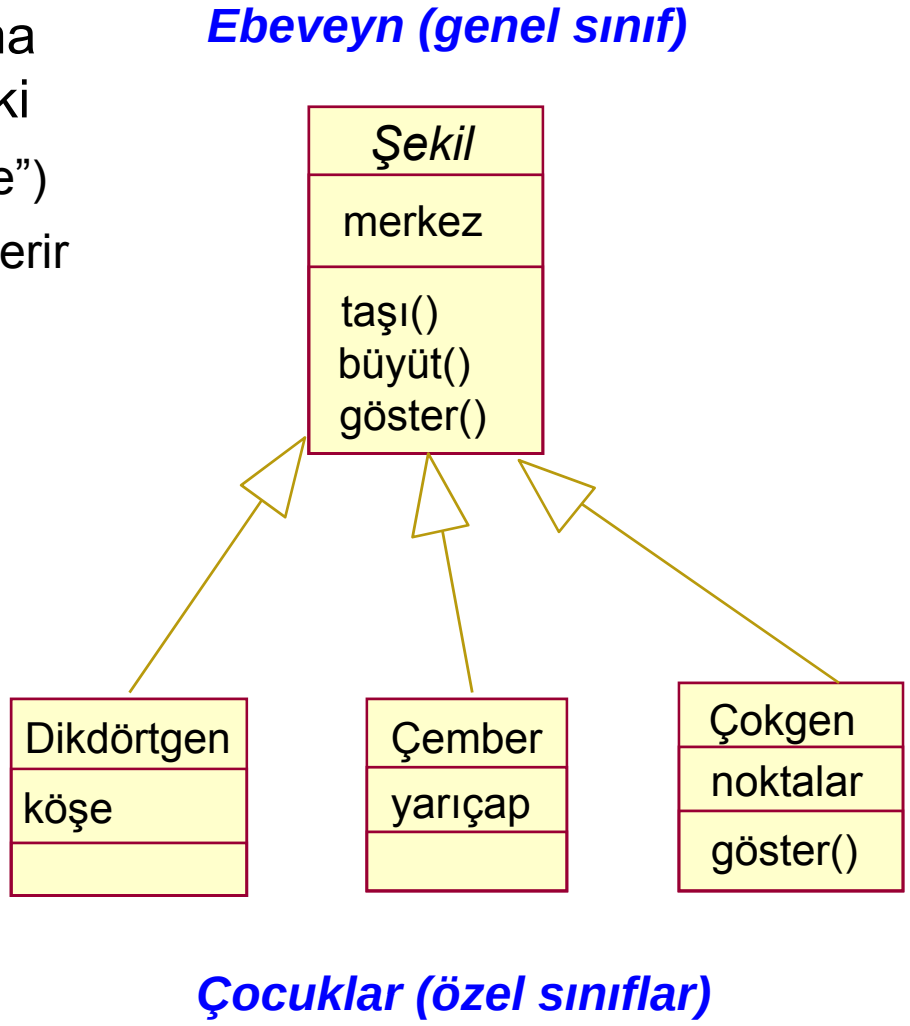
- Bir alt-sınıf, üst-sınıfından taşıdığı özelliklere ve işlevlere ek olarak; kendine ait özellikleri ve işlevleri içerebilir (tanımlayabilir).
  - Örnek: Dağ bisikleti, tırmanmayı kolaylaştıran ek viteslere sahip olabilir.
- Bir alt-sınıf aynı zamanda, üst-sınıfından taşıdığı işlevleri değiştirebilir (üzerine yazma – “method overriding”).
  - Örnek: Dağ bisikleti, bisiklet sınıfının “vites değiştir” işlevini, ek vitesleri kullanmayı sağlayacak şekilde değiştirebilir.
- Kalıtım sadece tek seviyeli olmak zorunda değildir, birden çok seviyede tanımlanabilir.
  - Bir alt-sınıf her zaman, üstündeki tüm sınıfların özelliklerini ve işlevlerini taşır.
  - Kalıtım ağacında (“inheritance tree”) aşağılara doğru inildikçe sınıfın öznelliği artar.

Kalıtım (“Inheritance”) – 3

- Sınıflar arasındaki kalıtım, uygulamada aşağıdaki avantajları sağlar:
  - Alt-sınıflar, üst-sınıflarının özelliklerini ve işlevlerini taşıdıklarından; programlama sırasında üst-sınıfların kodu defalarca tekrar kullanılabilir (“reuse”).
  - Programlama sırasında genel davranışları gösteren sınıflar *soyut sınıf* (“abstract class”) olarak kodlanabilir.
    - Soyutlama (“abstraction”)
    - Soyut sınıflardan nesne oluşturulmaz.
    - Soyut sınıflar, ilgili alt-sınıfları tanımlamak ve onlara ilişkin detayları doldurmak amacıyla kullanılırlar (tekrar kullanıma esas olarak).

Sınıflar Arasında Kalıtım İlişkisi (UML'de "Generalization")

- Genel sınıf ile onun özel durumlarına karşılık gelen sınıflar arasındaki ilişki
  - Ebeveyn-çocuk ilişkisi ("inheritance")
  - (UML) Okun yönü genel sınıfı gösterir
- Özel sınıflar genel sınıftan kalıtsal olarak özellikleri ve operasyonları alırlar.
- Özel sınıflar yeni özellikler ve operasyonlar tanımlayabilir veya kalıtsal yoldan aldıkları operasyonları yeniden tanımlayabilirler ("overriding").



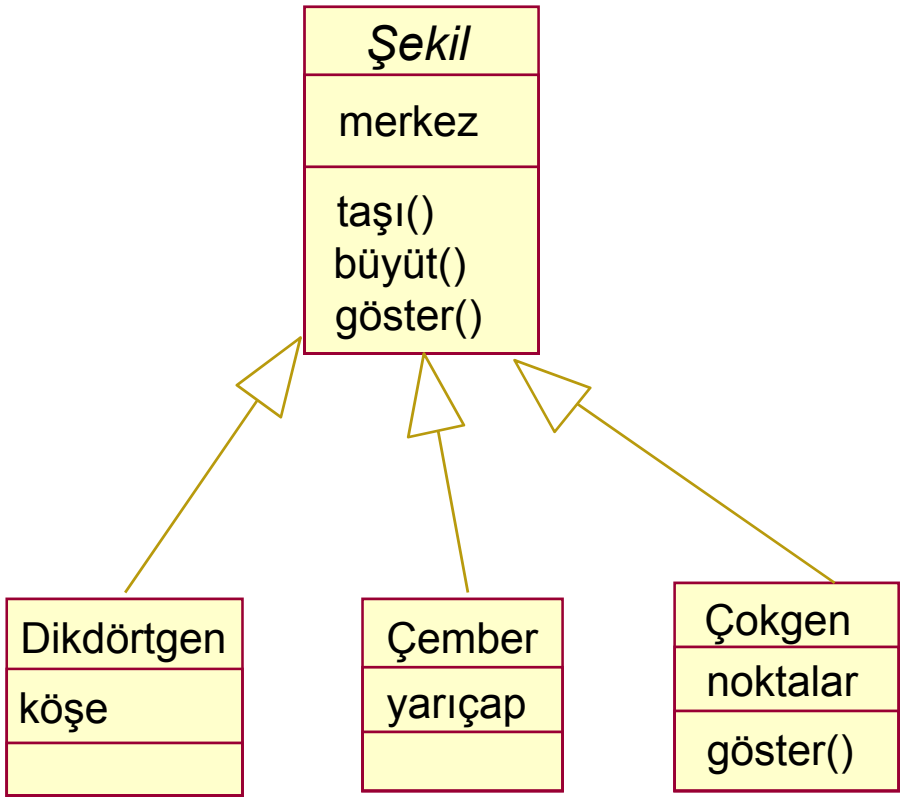
Java'da Kalıtım: Örnek - 1

```
class Sekil {
    …
    }

class Dikdortgen extends Sekil {
    …
    }

class Cember extends Sekil {
    …
    }

class Cokgen extends Sekil {
    …
    }
```



Java'da Kalıtım: Örnek – 2.1

```
class A {
    int i, j;
    void ijGoster () {
        System.out.println ("i ve j: "+ i + " " + j ); }
    }

class B extends A {
    int k;
    void kGoster () {
        System.out.println ("k: " + k);
        }
    void topla () {
        System.out.println ("i+j+k: " + (i+j+k));
        }
    }
```

Java'da Kalıtım: Örnek – 2.2

```
class SimpleInheritanceDemo {
   public static void main (String args[]) {
         A ustNesne = new A();
         B altNesne = new B();
         ustNesne.i = 10;
         ustNesne.j = 20;
         System.out.println("ustNesne içeriği:");
         ustNesne.ijGoster();
         System.out.println(); // bos satir yaz
         altNesne.i = 7;
         altNesne.j = 8;
         altNesne.k = 9;
         System.out.println("altNesne içeriği:");
         altNesne.ijGoster();
         altNesne.kGoster();
         System.out.println(); // bos satir yaz
         System.out.println("altNesne'de i, j, ve k toplami:");
         altNesne.topla();
         }
   }
```

Çıktı:

ustNesne içeriği:
i ve j: 10 20

altNesne içeriği:
i ve j: 7 8
k: 9

altNesne'de i, j ve k toplami:
i+j+k: 24

Sınıf Çalışması

- Farklı tipteki öğrencileri (lisans öğrencisi, y.lisans öğrencisi, vb.) göstermek için, bir Öğrenci sınıfını ve ilişkili alt sınıfları Java'da tanımlayın.
  - Özelliklerini gösterin.
  - Yöntemlerini gösterin.

Üzerine Yazma (“Method Overriding”)

- Bir alt sınıfta, üst sınıfa ait bir yöntemi; aynı isim, imza ve dönüş tipi ile tanımlarsak, üst sınıftaki yöntemin üzerine yazmış oluruz.
  - Alt sınıftan nesne oluşturulduğunda yöntem çağrılırsa, üst sınıfa ait yöntem yerine, alt sınıfta tanımlanmış yöntem koşturulur.

```
class Ogrenci {
    protected String ad;
    public Ogrenci (String pAd) {
        ad = pAd;
        }
    public String bilgiAl() {
        return “Ogrenci: ” + ad;
        }
    }
```

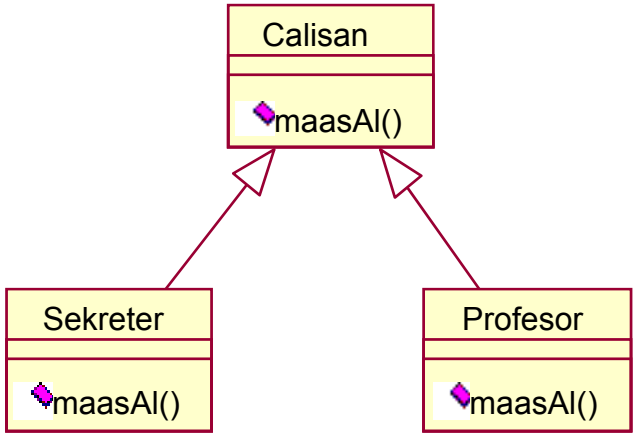
```
class LisansOgrencisi extends Ogrenci {
    public LisansOgrencisi (String pAd) {
        super(pAd);
        }
    public String bilgiAl() {
        return “Lisans ogrencisi: ” + ad;
        }
    }
```

Neden “Üzerine Yazma” ? - 1

- Genel sınıfta, kendinden türetilen tüm sınıflarda ortak olan işlevselliği tanımlamayı sağlar.

- Bir üst sınıftan alt sınıflara uzanan hiyerarşiyi tanımlamanın amacı, daha az detaydan daha çok detaya doğru işlevselliği oluşturmaktır.
  - Bu hiyerarşide üst sınıfın görevi, alt-sınıfların doğrudan kullanabilecekleri (veya üzerine yazabilecekleri) genel özellikleri ve yöntemleri tanımlamaktır.

Neden “Üzerine Yazma” ? - 2



Nesneye yönelik programlama yaparken koşulsal komutların mümkün olduğunca az, yöntem üzerine yazmanın (“overriding”) çok kullanılması önerilir.

- Üst sınıf aynı zamanda, alt sınıfları için tutarlı bir arayüz oluşturur (ortak tip)
- Bu sınıfları kullanan programlar, alt sınıflardan oluşturulan nesnelerin yöntemlerini, üst sınıfın yöntemlerini kullanır gibi kullanabilirler. Hangi seviyedeki sınıfın yönteminin kullanılacağına koşturma zamanında karar verilebilir (“polymorphism”).
  - Bu özellik, “if” veya “switch” kullanımına gerek bırakmaz. Yeni bir çalışan alt sınıfı eklendiğinde mevcut kodun değiştirilmesi gerekmez.

Kalıtım (“Inheritance”) İçin Öneriler

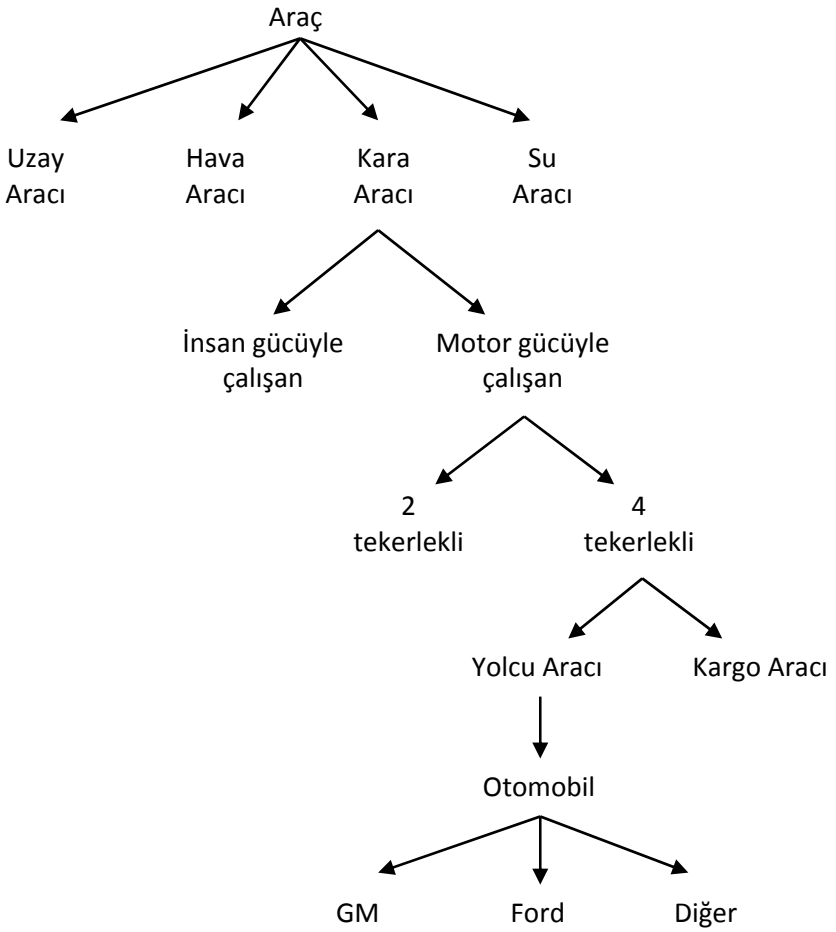
- Kalıtım ağacında (“inheritance tree”) diplerde yer almak, miras alınan operasyon sayısını ve karmaşıklığı arttırır.
  - Orta büyüklükte 100 sınıflı bir proje için 7 ± 2 seviyeli miras ağacı yeterli olacaktır.
  - Çok derin veya çok sığ yapılar kötü tasarım göstergesidir.

- Çok sayıda IF-THEN-ELSE ve SWITCH kullanımı, operasyonun ait olduğu sınıfın kötü tasarlandığını ve “inheritance” yoluyla aynı işin daha iyi yapılabileceğini gösterir.

- Miras ağacının yukarısındaki sınıflar daha aşağıdaki sınıflara bağımlı olmamalıdır.

Ödev - 2

- Sağda gördüğünüz araç türleri ağacını örnek alarak, tek seviyeli kalıtım sağlayan 3 araç seçin.
  - Bir üst sınıf, iki alt sınıf
- Sınıfların özellik ve yöntemlerini de gösteren kalıtım ağacını oluşturun (UML gösterimi ile).
- Sınıfları Java'da kodlayın.
  - Üst ve alt sınıflar arasında yöntemin üzerine yazmayı ("method overriding") örnekleyin.
  - Özelliklerin değerlerini yazdıran yöntemler tanımlayabilirsiniz.
- Sınıfları oluşturan ayrı bir demo sınıfı yazarak sınıflardan nesneler oluşturun ve yöntemlerini çağırarak özellikleri konsolda görüntüleyin.



***Teslim:* 24.Kasım.2009 24:00'e kadar atarhan@hacettepe.edu.tr adresine e-posta ile**