Inheritance

Computer Engineering Department

Java Course

Asst. Prof. Dr. Ahmet Sayar

Kocaeli University - Fall 2014

Kalıtım Mantığı

- 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 Mantığı -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 Mantığı -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”).
- Java’da (C++’da desteklenen) çoklu kalıtım (multiple inheritance) desteklenmez.
- Bir subclass’ın ancak bir tane direct super class’ı olabilir.
- Çoklu kalıtım yerine çoklu arayüz (multiple interface) kullanımı mevcuttur.
- Üzerine yazma/çeşitleme (overriding) sayesinde türeyen sınıflarda miras alınan metodlar ihtiyaçlara göre değiştirilebilir.

Kalıtım Mantığı -4

- Bir sınıfın diğerindeki özellikleri miras olarak alması için kullanılan anahtar sözcük ‘**extends‘** dir.
- Anlam olarak "Bu sınıf şu sınıfı genişletir” yani “ondaki özellik ve metodları alır ve yenilerini ekler“ demektir.
- Miras alınan sınıf - Superclass
- Miras alan sınıf – Subclass
- Herbir subclass, superclass olma adayıdır.
- Bir sınıftan türeyen sınıfın yapılandırıcısı türediği sınıfın yapılandırıcısını '**super()**‘ şeklinde çağırabilir.

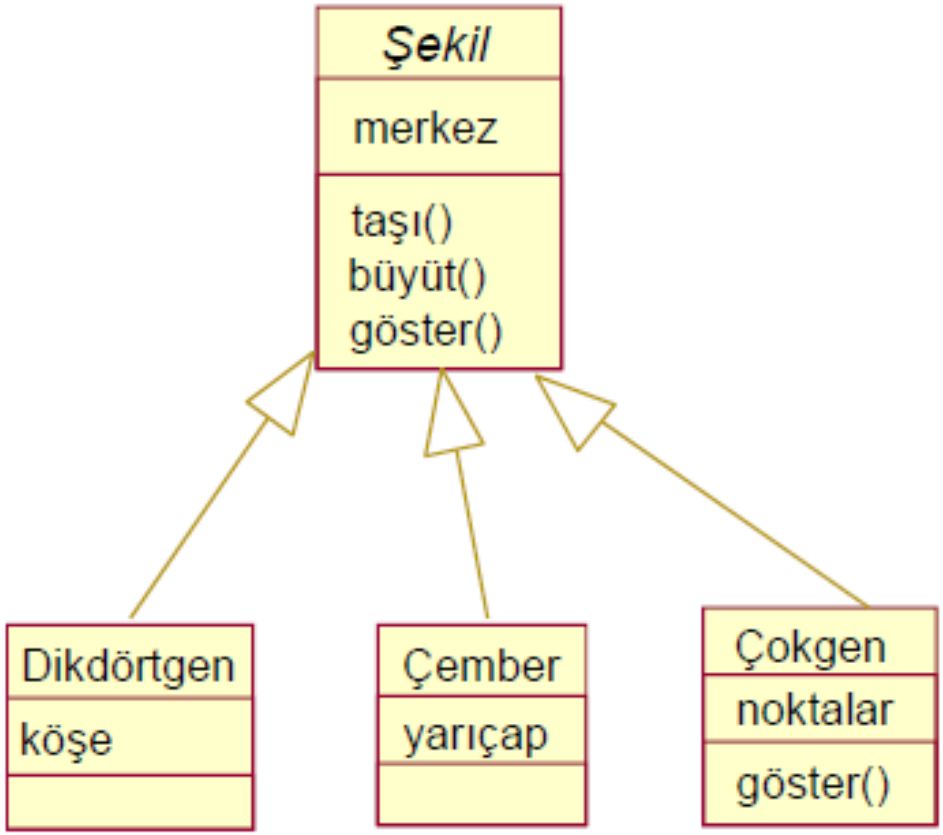
Java'da Kalıtım: Örnek - 1

```
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();
         }
   }
```

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;
        }
}
```

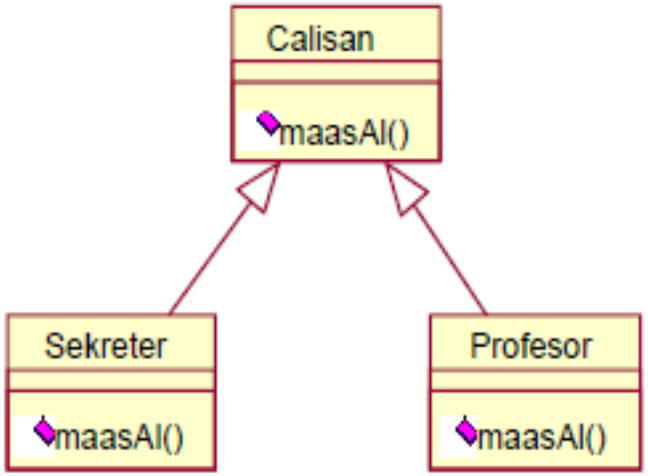
```
class LisansOgrencisi extends Ogrenci {
    public LisansOgrencisi (String pAd) {
        super(pAd);
        }
    public String bilgiAl() {
        return “Lisans ogrencisi: ” + ad;
        }
}
```

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.

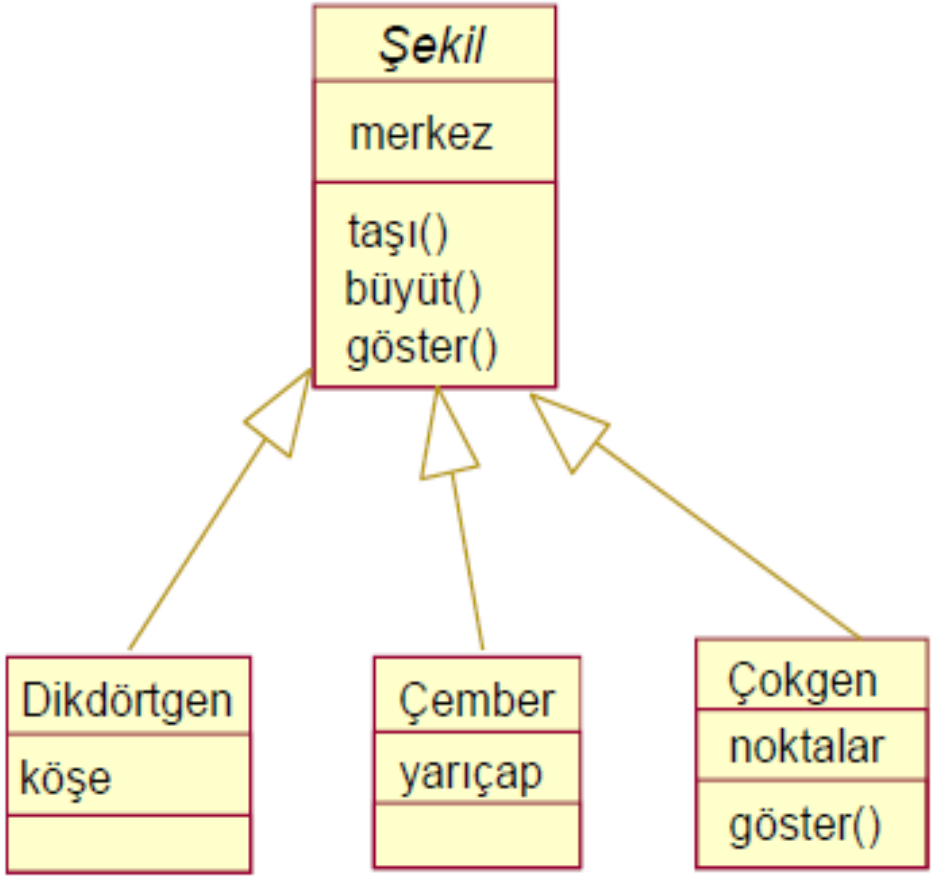
- Üst sınıf aynı zamanda, alt sınıfları için tutarlı bir arayüz oluşturur (ortak tip)
- Bu sınıfları kullanan programlar, alt sınıflardan oluşturulan nesnelerin yöntemlerini, üst sınıfın yöntemlerini kullanır gibi kullanabilirler. Hangi seviyedeki sınıfın yönteminin kullanılacağına koşturma zamanında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

Override edilmis metodu cagirma

- Super terimi kullanarak yapilir
- Super.maasAl();

Sınıflar Arasında Kalıtım İlişkisi

- Genel sınıf ile onun özel durumlarına karşılık gelen 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**”).



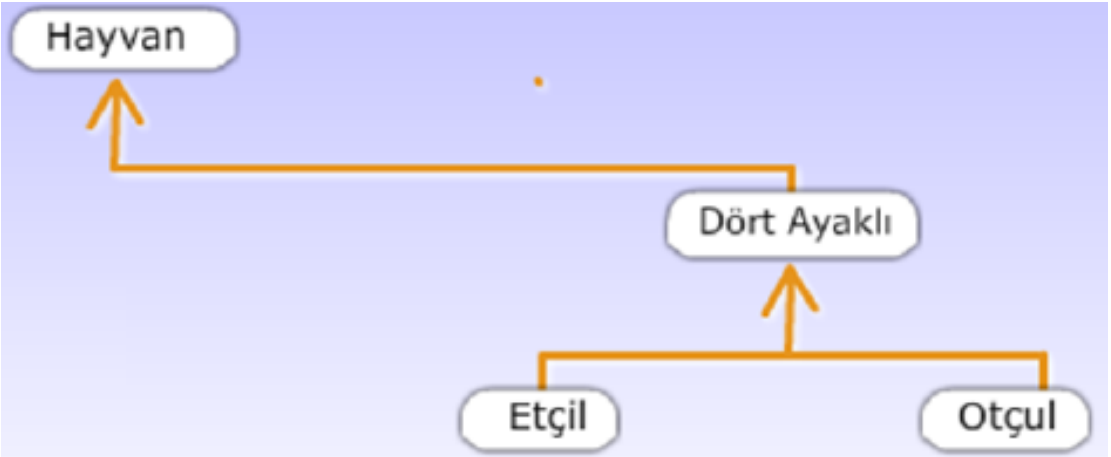
Kalıtım Örnek - I

Yapılandırıcılar

```
class Hayvan
{
   public Hayvan()
   {
      System.out.println("Hayvan SINIFI
YAPICISI");
   }
}
class DortAyakli extends Hayvan
{
   public Daire()
   {
      System.out.println("DortAyakli
SINIFI YAPICISI");

   }
}
class Etcil extends DortAyakli
{
   public Etcil ()
   {
      System.out.println("Etcil SINIFI
YAPICISI");
   }
}
class Otcul extends DortAyakli
{
   public Otcul ()
   {

      System.out.println("Otcul SINIFI
YAPICISI");
   }
}
public class Program
{
   public static void main(String[] args)
   {
      Otcul o=new Otcul();
   }
}
```



EKRAN ÇIKTISI

Hayvan SINIFI YAPICISI
DortAyakli SINIFI YAPICISI
Otcul SINIFI YAPICISI

```
Class Sekil
{
    double Taban;
    double Yukseklik;
    public Sekil(double t,double y)
    {
        Taban=t;
        Yukseklik=y;
    }
}

class Ucgen extends Sekil
{
   String UcgenTuru;
   public Ucgen(double x,double y,String
tip)
    {
       super(x,y);
   UcgenTuru=tip;
    }
   public double Alan()
    {
       return (Taban*Yukseklik)/2;
    }
}
public class Program
{
    public static void main(String[] args)
    {
        Ucgen u=new
 Ucgen(8,18,"Eskenar");
        System.out.println("Ucgen alani
 "+u.Alan());
    }
}
```

Kalıtım Örnek-II

- Ucgen sınıfından bir nesne 3 parametre alan yapıcı metodu ile oluşturulduğunda ilk 2 parametre ana sınıftaki yapılandırıcıya aktarılmış, böylece Sekil sınıfından türetilen nesnelerin ortak özelliği olan Taban ve Yuksekliği türeyen sınıfın içinde tekrar bildirme zorunluluğu ortadan aldırılmıştır.

Kalıtım Örnek-III

```
public abstract class Cat {
        public void greet() {
                System.out.println("Meow");
        }
}
public class Tomcat extends Cat {
        public void greet(Cat c) {
                System.out.println("What up fool?");
                c.greet();
        }
}
public class Siamese extends Cat {
        public void greet() {
                System.out.println("Meeeeow!");
        }
        public void greet(Cat c) {
                System.out.println("Hellow");
        }
}
```

Asagidaki kodlarin ciktilari nedir?

a. (new Tomcat()).greet();
b. (new Tomcat()).greet(new Siamese());
c. Cat c = new Siamese();
((Cat) c).greet();

d. Cat c = new Siamese();
((Tomcat) c).greet(c);

Kalıtım Örnek-IV

```
public class FamilyTester {
        public static void main(String[] args) {
                Grandparent gramps = new Parent();
                gramps.whoAmI();
                Parent papa = new Child();
                papa.whoAmI();
                Child sonny = new Child();
                sonny.whoAmI();
        }
}
public class Grandparent {
        public void whoAmI() {
                System.out.println("I'm a Grandparent");
        }
}
public class Parent extends Grandparent {
        public void whoAmI() {
                System.out.println("I'm a Parent");
        }
}
public class Child extends Parent {
        public void whoAmI() {
                System.out.println("I'm a Child");
        }
}
```

Yandaki main yordaminin ciktisi nedir?

Programin Ciktisi:

I'm a Parent
I'm a Child
I'm a Child

Final Deyimi

- Nesne yönelimli programlamanın getirdiği yeni kavramlarla birlikte metodlar ve sınıflar içinde değişmeyen yapılar tanımlama ihtiyacı doğmuştur.
- Java'da değişkenlerin, metodların ve sınıfların özelliklerinin değişikliğe uğramasını engellemek için bu yapılara final niteliği atanır.
  - Final değişkenler
  - Final sınıflar
  - Final metodlar

Final değişkenler

- Bir değişken final niteliği ile tanımlanırsa, ilk değeri atandıktan sonra bir daha değeri değiştirilemez.
- Final değişkenlere ilk tanımlandığı anda değerinin atanması gerekmektedir.
- Örn. final float pi=3.14;

  final static tbmm = “Turkiye Buyuk Millet Meclisi”;

Final sınıflar

- Nesneye yönelimli programlamanın en önemli özelliklerinden birisi bir sınıfın niteliklerini yeni bir sınıfa aktarabilmektir.
- Bu yönteme kalıtım denilmektedir.
- Ancak, bir sınıf final deyimiyle oluşturulduğunda bu sınıftan yeni sınıflar türetilmesi engellenmiş olur.

Final metodlar

- Kalıtım yöntemiyle bir sınıfın özellikleri ve işlevleri yeni sınıflara aktarılabildiği gibi yeni sınıfta metodların üzerinde değişiklik yapabilmektedir.

- Eğer bir metodun, türeyen sınıflarda değiştirilmesi istenmiyorsa final deyimi kullanılarak bu engellenebilir.

finalize() metodu

- Bazı programlama dillerinde nesneyi sonlandırmak için imha düzenekleri (destructor) mevcuttur.
- Bu yapılar genellikle belleğin temizlenmesi, diğer nesnelerle ilişkilerin sonlandırılması, açık dosyaların kapatılması gibi işlemleri içermektedir.
- Java nesneyi yoketmek için çöp toplayıcı mekanizması kullandığı için ayrıca bir imha düzeneğine ihtiyaç yoktur. Ancak, yine de programcının nesne sistemden silinirken yapılmasını istediği işlemler olursa bunları yazabilmesi için finalize() metodu kullanılmaktadır.

```
public class Kitap{

.............

        public void finalize ( ){
                System.out.println("SON");
        }
}
```

“Is A” ve “Has a” Iliskileri

- IS A:
  - Turetilmis sinif hem kendi tipindedir hem de turetildigi sinif tipindedir.
    - Ornek, ogrenci insan sinifindan turetilmis olsun.
    - Ogrenci sinfi ornegi Ali ayni zamanda bir insandir.
- HAS A:
  - Bir sinif baska bir sinifi oz niteligi olarak sahip olabilir.
    - Ornek, Ogrenci sinifi ve Ders sinifi olsun
    - Ogrenci sinifi icinde Ders[] dizisi oz nitelik olarak tanimlanmis olabilir.

“Object” (Nesne) Sinifi

- JAVA da tüm sınıflar Object sınıfından türer.
  - Yazılsa da yazılmasa da
- Bu nedenle tüm sınıfların tüm örnekleri nesne tipindeddir.
- Tüm sınıflar Object sınıfının aşağıdaki metodlarını kalıtır
  - equals()
  - toString()
  - clone()
- toString() sınıfı genellikle override edilir.
- Clone objenın kopyasını olusturur. Diziler konusundan hatırlayın.