



Kapsülleme (“Encapsulation”)

BBS-515 Nesneye Yönelik Programlama

Ders #5 (18 Kasım 2009)

■ Geçen ders:

- ▶ Kalıtım (“inheritance”)

■ Bu ders:

- ▶ Kalıtım (“inheritance”) – tekrar ziyaret
- ▶ Java’da “super” kullanımı
- ▶ Java’da “static” ve “final” tanımları
- ▶ Kapsülleme (“encapsulation”)



Kalıtım (“Inheritance”) – Tekrar Ziyaret

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*

Örnek – 3

```
public class Bisiklet {  
  
    public int vites;  
    public int hiz;  
  
    // olusturucu metod  
    public Bisiklet(int pHiz, int pVites) {  
        this.vites = pVites;  
        this.hiz = pHiz;  
    }  
  
    // diger metodlar  
    public void vitesDegistir(int yeniDeger) {  
        this.vites = yeniDeger;  
    }  
  
    public void fren(int pAralik) {  
        this.hiz -= pAralik;  
    }  
  
    public void hizlan(int pAralik) {  
        this.hiz += pAralik;  
    }  
  
}
```

```
public class DagBisikleti extends Bisiklet {  
  
    public int koltukYuksekligi;  
  
    // olusturucu metod  
    public DagBisikleti(int pYukseklik, int pHiz, int pVites) {  
        super(pHiz, pVites);  
        this.koltukYuksekligi = pYukseklik;  
    }  
  
    // eklenen metod  
    public void koltukYuksekliginiDegistir(int yeniYukseklik) {  
        this.koltukYuksekligi = yeniYukseklik;  
    }  
  
}
```



Java'da "super" Kullanımı

“super” Kullanımı

- “super” sözcüğünün iki tür kullanımı vardır:
 - ▶ Üst sınıfın oluşturucusunu (“constructor”) çağırmak için kullanılır.
 - ▶ Üst sınıfın bir elemanına (özeğe veya yöntem) ulaşmak için kullanılır.

“super” Kullanımı: Oluşturucuyu Çağırarak

```
class Kutu {  
    Tümü özel tanımlı {  
        private double genislik;  
        private double yukseklik;  
        private double derinlik;  
    }  
    Kutu (Kutu ob) {  
        genislik = ob.genislik;  
        yukseklik = ob.yuksekklik;  
        derinlik = ob.derinlik;  
    }  
    Kutu (double g, double y, double d) {  
        genislik = g;  
        yukseklik = y;  
        derinlik = d;  
    }  
    Kutu () {  
        genislik = -1;  
        yukseklik = -1;  
        derinlik = -1;  
    }  
    .....  
}
```

```
class AgirKutu extends Kutu {  
    double agirlik;  
    AgirKutu (AgirKutu ob) {  
        super (ob);  
        agirlik = ob.agirlik;  
    }  
    AgirKutu (double g, double y, double d, double a){  
        super (g,y,d);  
        agirlik = a;  
    }  
    AgirKutu () {  
        super();  
        agirlik = -1;  
    }  
}
```

“super” Kullanımı: Üst Sınıfın Bir Elemanına Ulaşmak

(Genellikle üst sınıf ve alt sınıfta özellik ve yöntem isimleri aynı ise kullanılır.)

```
class A {
    int i;
}

class B extends A {
    int i;           // A'dakinin üzerine yazar

    B (int a, int b) {
        super.i = a; // A'da tanımlı i
        i = b;       // B'nin kendi özelliği olan i
    }

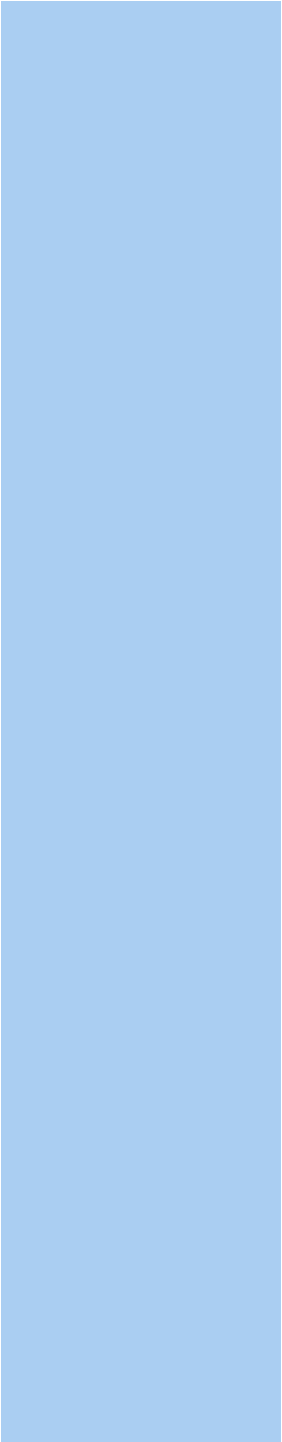
    void goster () {
        System.out.println("Üst sınıftaki i: " + super.i);
        System.out.println("Alt sınıftaki i: " + i);
    }
}

class SuperDemo {
    public static void main(String args []) {
        B altOb = new B (1,2);
        altOb.goster();
    }
}
```

ÇIKTI:

Üst sınıftaki i: 1

Alt sınıftaki i: 2



Java'da “static” ve “final” Tanımları

“static” Tanımı - 1

- Bir sınıfın nesnelere bağımsız olarak kullanılacak özellik veya yöntemi tanımlarken kullanılır.
- Sınıfın bir elemanı (özelligi veya yöntemi) “static” tanımlandıysa, o elemana sınıftan nesne oluşturmadan ulaşılabilir.
 - ▶ *Sinifadi.ozellik*
 - ▶ *Sinifadi.yontem()*
- Örnek: *public static void main ()*

“static” Tanımı - 2

- Bir sınıfın “static” olarak tanımlanmış değişkenleri genel kapsamlıdır. O sınıfın tüm nesnelere aynı değişkeni kullanır.
- Bir sınıfın “static” olarak tanımlanmış yöntemleri, ancak diğer “static” yöntemleri çağırabilir ve “static” veriye ulaşabilir; “this” ve “super” sözcüklerini kullanamaz.
 - ▶ **main() bloğu hariç**
- “static” değişkenler ancak “static” tanımlı bir blok tarafından ilklendirilebilir.
- “static” tanımlı bir blok yalnız bir kez işletilir.

“static” Tanımı: Örnek

```
class Statik1 {
    static int a=3;
    static int b;
    static void met (int x) {
        System.out.println (“x= ”+ x);
        System.out.println (“a= ”+ a);
        System.out.println (“b= ”+ b);
    }
    static {
        System.out.println
        (“Statik blok işletildi”);
        b= a*4;
    }
    public static void main (String args []) {
        met (42);
    }
}
```

Bir kez işletilir.

Çıktı:

Statik blok işletildi.

x=42

a=3

b=12

```
class Statik2 {
    static int a = 42;
    static int b = 99;
    static void beniCagir () {
        System.out.println(“a = ” +a);
    }
}
class Statik2Demo {
    public static void main (String args []) {
        Statik2.beniCagir();
        System.out.println(“b = ” +
        Statik2.b);
    }
}
```

Çıktı:

a=42

b=99

“final” Tanımı - 1

- C/C++ programlamada sabit tanımlamaya benzer.
- “final” olarak tanımlı bir değişkene yalnız bir kez değer atanabilir
 - ▶ Değişkenin değeri, bir kere atandıktan sonra değiştirilemez.
- Örnek:
 - ▶ `final int FILE_NEW = 1;`
 - ▶ `final int FILE_OPEN = 2;`
- “final” tanımlı değişkenleri kodlarken büyük harf kullanılması önerilir.

“final” Tanımı - 2

- Kalıtım kullanırken yöntemin üzerine yazmayı engellemek için kullanılır.

- Örnek:

```
class A {  
    final void met() {  
        System.out.println (“Bu metot final tanımlanmıştır.”);  
    }  
}
```

```
class B extends A {  
    void met() { // Hata: Geçersiz yöntem tanımı  
    }  
}
```

“final” Tanımı - 3

- Kalıtımı engellemek için de kullanılır.

- Örnek:

```
final class A {  
    // ...  
}
```

```
class B extends A {           // Hata: Geçersiz sınıf tanımı  
}
```



Kapsülleme (“Encapsulation”)

Kapsülleme (“Encapsulation”) - 1

- Bir sınıfın özellikleri ve işlevleri o sınıfa aittir ve detayları dışarıdan bilinmez.
- Özelliklere ve işlevlere sınıf izin verdiği ölçüde dışarıdan ulaşılır.
 - ▶ Erişim için özellikler ve işlevler “public”, “protected”, “private” olarak tanımlanır.
- Dışarıdan ulaşan bir diğer sınıf, sadece sınıfın işlevini nasıl çağıracağını bilir; işlevin nasıl gerçekleştirildiğini bilmez (“encapsulated implementation”)
 - ▶ Örnek: Araba motorunun çalışması (sürücü sadece anahtarı çevirir ve motor çalışır; motorun nasıl çalıştığı sürücüden gizlidir.)

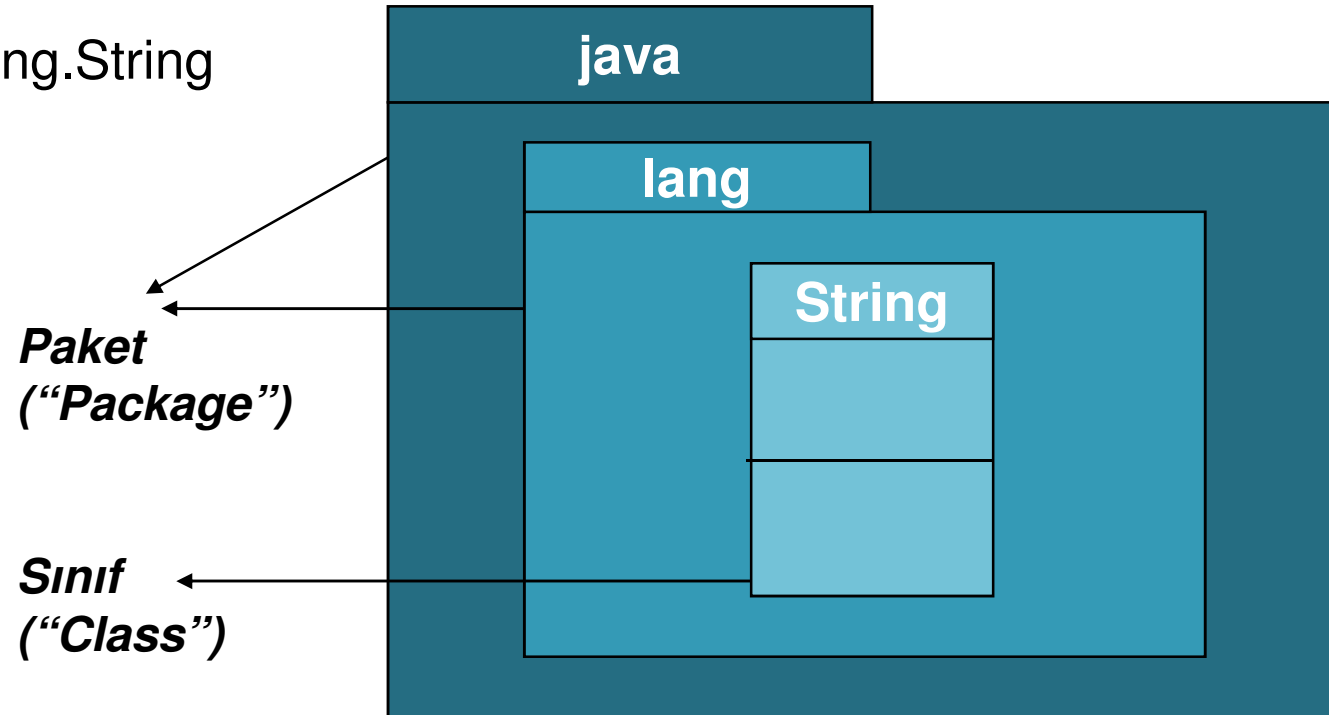
Kapsülleme (“Encapsulation”) - 2

- Kapsülleme özelliği aşağıdaki avantajları beraberinde getirir:
 - ▶ Modülerlik: Bir nesnenin kaynak kodu diğerlerinden bağımsız olarak yazılıp yönetilebilir.
 - ▶ Bilgi saklama: Bir nesne, kendisiyle iletişimde bulunan diğer nesnelere etkilemeden, sadece kendinin ulaşabileceği özellik ve yöntemlerini değiştirebilir.

Paket (“Package”) Kavramı

- Java’da sınıf isimleri birbirinden farklı olmalıdır. Adres uzayında sınıf isimlerini tek (“unique”) yapmak için, paket (“package”) kavramı tanımlanmıştır.

- Örnek: `java.lang.String`



Eriřim Kontrolü - 1

- Kapsülleme (“encapsulation”) özelliđi, veriyi onu deđiřtiren kodla iliřkilendirir.
 - ▶ İyi tanımlı arayüzleri olan kara-kutular oluřturmayı hedefleriz.
 - ▶ Kara-kutuların iç iřleyiři dıřarıdan müdahaleye açık olmamalıdır.
- Bu řekilde bir nesne ancak kendi özelliklerinin deđerlerini deđerştirebilir.
- Bir sınıfın özelliklerine ve yöntemlerine eriřim hakkını tanımlamak için eriřim belirticileri (“access specifiers”) kullanırız.

Erişim Kontrolü - 2

- Java'da kullanılan erişim belirteciler (“access specifiers”) şunlardır:
 - ▶ **public**: Özellik veya yöntem, program içindeki herhangi bir koddan erişilebilir.
 - ▶ **private**: Özellik veya yöntem, ancak o sınıfın içindeki herhangi bir koddan erişilebilir.
 - ▶ **protected**: Özellik veya yöntem, alt-sınıfların içindeki veya aynı paketdeki başka bir sınıfın içindeki herhangi bir koddan erişilebilir.

- Ana yöntem için “public” erişim belirtici kullanılır.

- Örnekler:
 - ▶ `public int i; private double j;`
 - ▶ `private int met (int a) {.....}`

Erişim Belirticiler (“Access Specifiers”)

	Private	Belirtici Yok	Protected	Public
Aynı sınıf	Evet	Evet	Evet	Evet
Aynı paket ve alt-sınıf	Hayır	Evet	Evet	Evet
Aynı paket ve alt-sınıf değil	Hayır	Evet	Evet	Evet
Farklı paket ve alt-sınıf	Hayır	Hayır	Evet	Evet
Farklı paket ve alt-sınıf değil	Hayır	Hayır	Hayır	Evet

Özellik Yazma (“Setter”) ve Özellik Okuma (“Getter”) Yöntemleri

- Erişim belirtici (“access specifier”), özellik için erişilebilirliği belirler (public/protected/private) .
 - ▶ private int id; // diğer sınıflar göremez.
 - ▶ public String name; // diğer sınıflar görebilir.
- Bir sınıfın erişilemez özelliklerine ulaşmak için, özellik yazma (“setter”) ve özellik okuma (“getter”) yöntemleri kullanılır.
 - ▶ **Özellik yazma (“setter”) yöntemi:** Sınıfın erişilemez özelliğine dışarıdan yazmayı sağlar.
 - ▶ **Özellik okuma (“getter”) yöntemi:** Sınıfın erişilemez özelliğini dışarıdan okumayı sağlar.

Sınıf Çalışması

- Öğrenci sınıfını Java'da tanımlayın.
 - ▶ Özelliklerini farklı erişim belirtecilerle tanımlayın.
 - ▶ Özellik yazma (“setter”) ve özellik okuma (“getter”) yöntemlerini tanımlayın.
- Çalıştırılabilir başka bir sınıf içinde öğrenci nesnesi oluşturun. Nesnenin özelliklerine değerler atayın ve sonra değerleri okuyarak yazdırın.

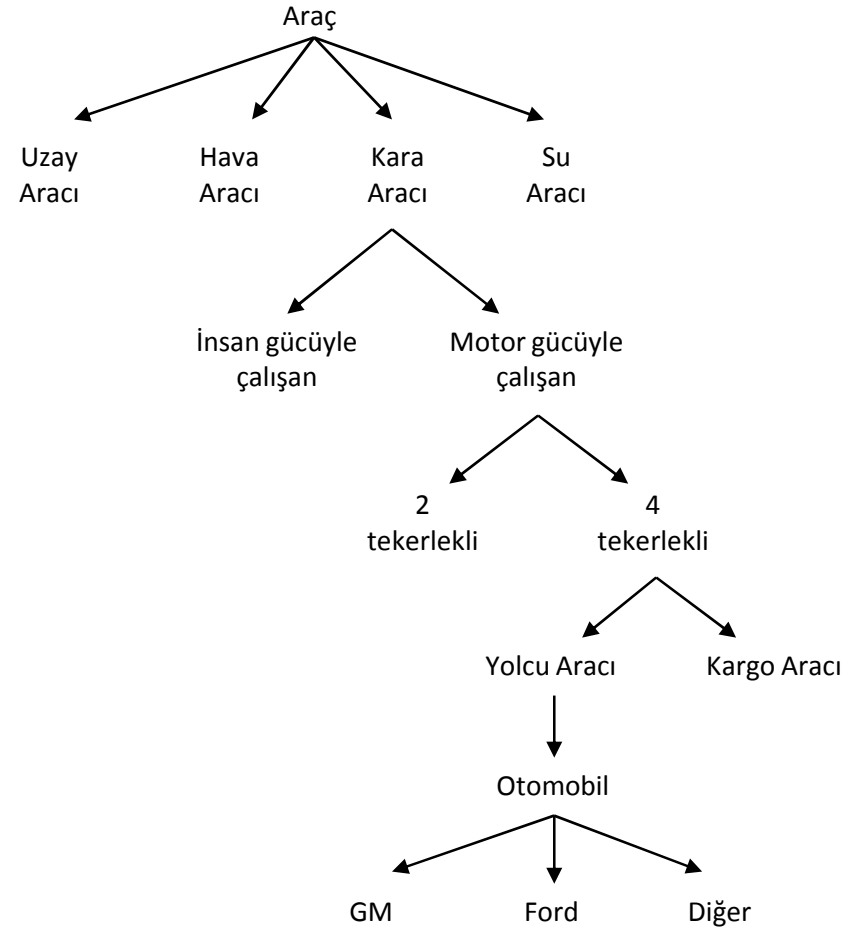
```
public class Ogrenci {
    private int no;
    public String ad, adres;

    public Ogrenci() {
        no = 0;
        ad = "";
        adres = "";
    }
    public void setNo(int pNo) {
        no = pNo;
    }
    public int getNo(){
        return this.no;
    }
    public void bilgiYaz(){
        System.out.println (no + "," +
        ad + "," + adres);
    }
}
```

```
public class Test {
    public static void main(String args[]) {
        // nesne degiskenini tanimla
        Ogrenci ogr;
        // nesneyi yarat
        ogr = new Ogrenci();
        // nesne özelliklerini ilklendir
        ogr.setNo(836);
        ogr.ad = "Ayşe";
        ogr.adres = "Ankara";
        // nesne bilgisini yazdir
        ogr.bilgiYaz();
        // nesne tanımlayiciyi yazdir
        System.out.println(ogr.getNo());
    }
}
```

Ödev – 3: Günleme

- Sağda gördüğünüz araç tür 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).
 - ▶ Özellik yazma (“setter”) ve özellik okuma (“getter”) yöntemlerini kullanın.
- Sınıfları Java’da kodlayın.
 - ▶ Üst ve alt sınıflar arasında yöntemin üzerine yazmayı (“method overriding”) örnekleyin.
 - ▶ Özelliklerin değerlerini yazdıran yöntemleri tanımlayabilirsiniz.
- Sınıfları oluşturan ayrı bir demo sınıfı yazarak sınıflardan nesnelere oluşturun ve yöntemlerini çağırarak özellikleri konsolda görüntüleyin.



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