

tart
yeni medya mutfağı



Nesneye Yönelik Programlama

Cihangir SAVAS | SW101

Seminer İerięi

01. Class
02. Eriřim Denetleyiciler
03. Method Overloading
04. Yapılandırıcılar
05. İlk deęer ataması
06. This
07. Static / Final
08. TO BE CONTINUED :)



Class Nedir?

Nesneler

- Dizi, aynı tipteki değerlerin bir koleksiyonudur.
- Nesne ise farklı tipteki değerlerin bir koleksiyonudur.
 - Aslında nesne bundan biraz daha karmaşıktır ama bu tanımı sadece karşılaştırma için yapıyoruz.
- Örnek:

```
class Person {  
    String name;  
    boolean male;  
    int age;  
    int phoneNumber;  
}
```
- Bu sebeple farklı tipteki nesneler bellekte farklı boyutta yer alacaktır.

Erişim Belirleyiciler

public: Her yerden erişilmeyi sağlayan erişim belirleyicisi.

protected: Aynı paket içerisinde ve bu sınıftan türemiş alt sınıflar tarafından erişilmeyi sağlayan erişim belirleyicisi.

private: Yalnızca kendi sınıfı içerisinde erişilmeyi sağlayan, başka her yerden erişimi kesen erişim belirleyicisi.

private & public alanlar ve metodlar

- **private** alanlar türetilmiş sınıflarda kullanılamazlar.
 - “Bilgi Gizleme” böyle olması gerektiğini söyler
 - Eğer erişmek gerekiyorsa üst sınıfın ilgili metodları kullanılarak gerekli erişim yapılır.
- **private** metodlarda alt sınıflara kalıtımsal olarak geçmezler.
 - Alt sınıflarda kullanılması gerekiyor ise `public/protected` olarak tanımlanmalıdır
 - Sadece yardımcı olmak için bazı metodlar `private` tanımlanabilir

protected Belirleyicisi

Görülebilir özellikler hangi alanların/metodların kalıtılabileceğini veya kalıtlamayacağını gösterirler public olarak tanımlanan değişkenler ve metodlar kalıtılabilirken, private olanlar görünmeyen özellikler olup kalıtılmamaktadır.

Fakat public alanlar/değişkenler kapsama ilkeline ters olan bir belirleyicidir.

Bu sebeple kalıtım durumunda yardımcı olan bir üçüncü belirleyici geliştirilmiştir : **protected**

protected Belirleyicisi – 2

protected belirleyicisi üst sınıfın bir üyesinin (alan veya metod) alt sınıfa geçmesini sağlar.

Protected şu iki özelliği sağlar

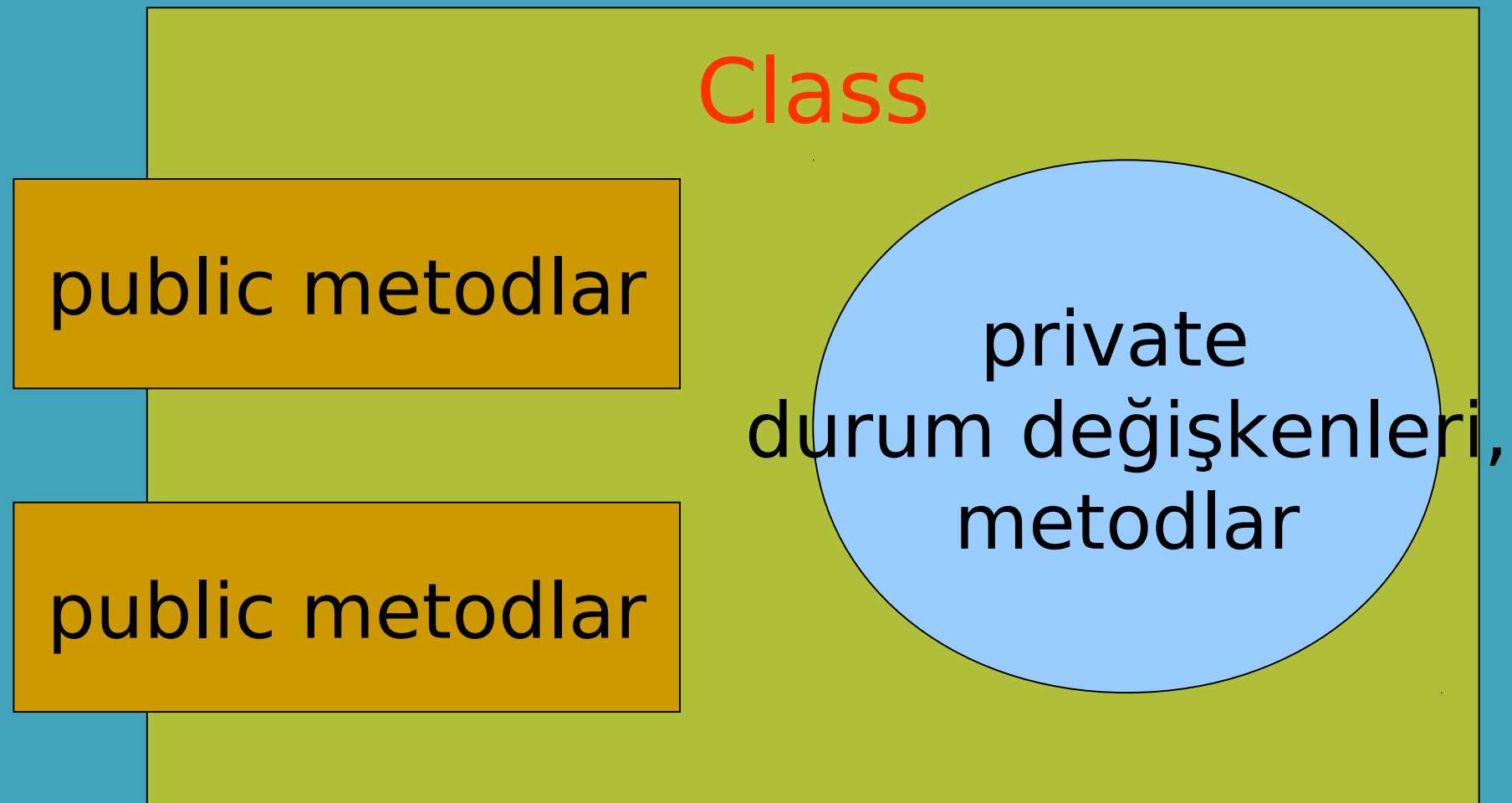
- public ten daha fazla bir kapsama özelliği vardır.
- Kalıtıma izin veren en iyi kapsama imkanını sağlar.

Tasarım Temelleri

- *private* durum değişkenleri kullanılarak özel veriler sınıf yapısı içinde sarmalanır-kapsülленir.
(Encapsulation)
- Sınıf dışından bu verilere erişim public metodlar kullanılarak yapılır.
- *private* metodlar sadece sınıf içinden erişim için kullanılabilir.

Sarmalama-Kapsülleme

Data Encapsulation



Sınıftaki private/public metodlar

```
private String isim, tel;  
private double haftalikmaas, yillikmaas;  
    // aynı sınıf içinden erişim için kullanılır  
private void HesaplaYillikMaas() {  
    yillikmaas = haftalikmaas * 12;  
}  
public void YazdirYillikMaas() {  
    HesaplaYillikMaas(); // aynı sınıf içinden erişim yapılıyor.  
    System.out.println(isim+" nin yıllık maasi " + yillikmaas);  
}  
}
```

public Metodların kullanımı

```
class test {  
    public static void main(String args[]) {  
        Calisan tart=new Calisan();  
        tart.YazdirYillikMaas(); //şeklinde kullanılır  
        tart.HesaplaYillikMaas(); //kullanılmaz  
    }  
}
```

Bilgisayarda Bellek Organizasyonu

- Bir **bit** on/off, yes/no, **0/1** tipinden değerdir
 - Bir **byte** ise sekiz ardışık bitten oluşur
 - Bilgisayar Belleği ise 0 dan başlayan bir dizi byte dan oluşur
 - Bellekteki herhangi bir adresin boş olması diye bir kavram yoktur burada bir şekilde bitler(0/1) saklıdır.
 - Bir byte ilişkilendirilmiş olan numara onun **adresidir**.
-
- Adresler ikili düzendeki sayılardır (genellikle hexadecimal olarak yazılırlar)
 - Bazı büyük bilgisayar sistemleri ise byteları değil wordleri adresler. **Wordler** ise daha fazla sayıda bitten oluşur (such as 32 or 64)

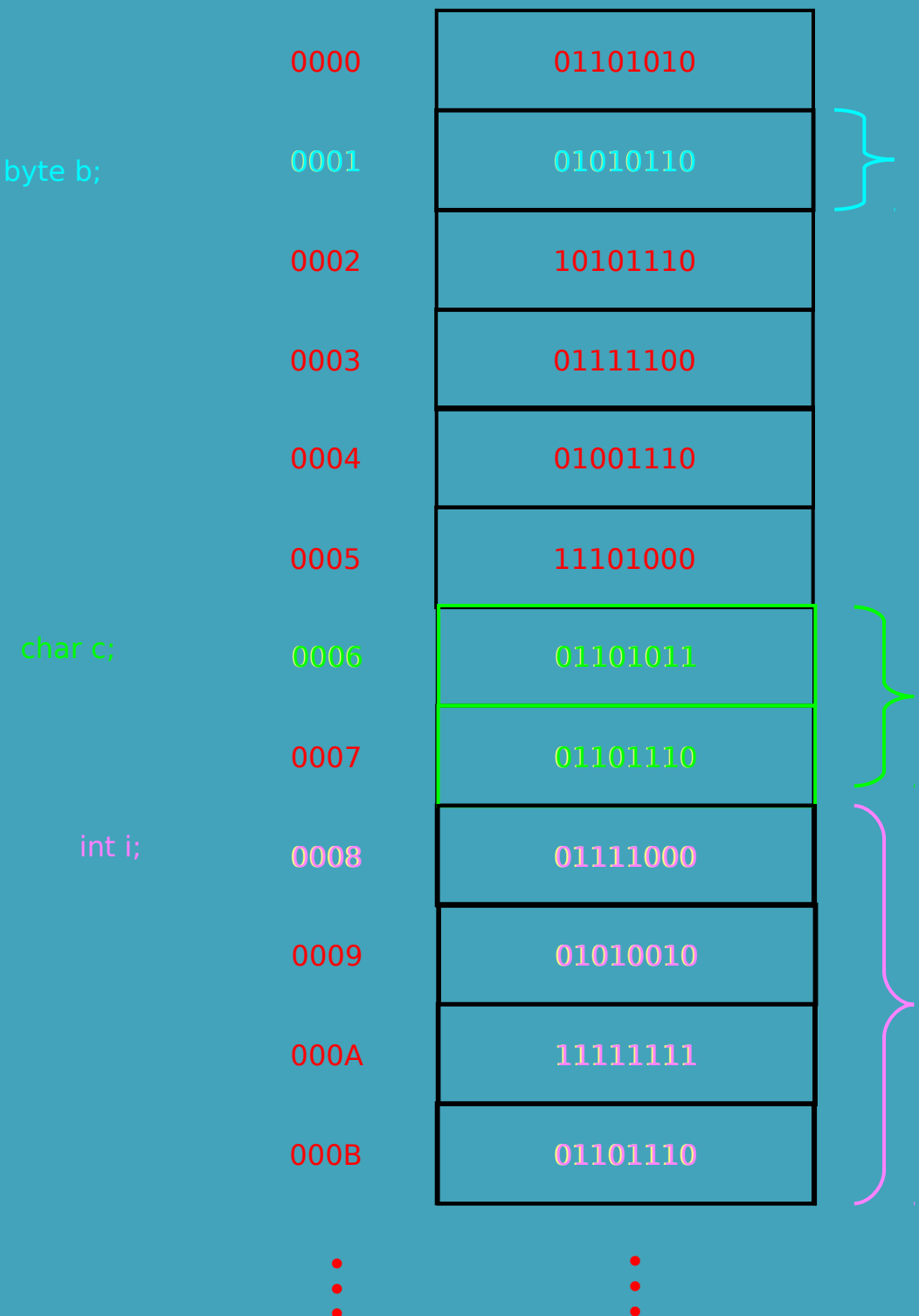
• 0000	• 001101010
• 0001	• 101010110
• 0002	• 10101110
• 0003	• 01111100
• 0004	• 01001110
• 0005	• 11101000
• 0006	• 01101011
• 0007	• 01101110
• 0008	• 01111000
• 0009	• 01010010
• 000A	• 11111111
• 000B	• 01101110

•
•
•

•
•
•

Bellekteki basit değişkenler

- Basit bir değişkeni tanımlayında bellekte ona bir yer ayırırsınız. (Tipine göre)
- Örneğin bir **byte** bellekte tek bir bytelık alana `byte b;` ihtiyaç duyar
- Bir **int** ise dört ardışık byte a ihtiyaç duyar
- Bir **char** iki ardışık byte a ihtiyaç duyar
- Bu yüzden her değişkenin bellekte ayrı bir adresi vardır.



Bellekte Nesne Saklama

- Bir nesneyi tanımladığımız zaman ise Derleyici bu nesnenin ne kadar belleğe ihtiyacı olduğunu bilemez. Bu sebeple bellek ataması yapamaz.
- Gerekli bellek miktarı o nesneyi yarattığınızda bellekte tanımlanır ve atanır.

0000	01101010
0001	01010110
0002	10101110
0003	01111100
0004	01001110
0005	11101000
0006	01101011
0007	01101110
0008	01111000
0009	01010010
000A	11111111
000B	01101110
⋮	⋮

Nesnenin Tanımlanması ve Yaratılması

Person p = new Person("John");

Person p işaretçisi için bellekte bir yer tanımlar.

new Person("John"), Person sınıfının yapısına uygun bellekte bir yer ayırır.

Atama komutu ile p ye gerekli referans ataması gerçekleşir.

p:

07A3

07A3

"John"

Atama İşlemleri

- Basit Değişkenlerde veri atamalarda *değer* ataması yapılır
- Nesnelerin atanmasında ise *referansların kopyalaması* yapılır.
- Bu sebeple iki nesne aynı veri bloğuna işaret edebilir.
 - Bunu başka bir isimle adlandırma olarakta düşünebilirsiniz.
 - Eğer bir nesnenin işaret ettiği bir değeri değiştirirseniz diğer nesnede değişecektir.

Parametrelerde Veri transferi

- Bir method çağrısı şu şekilde olur.

```
result = add( 3 , 5 ) ;
```

Parametre değerleri kopyalanarak gönderilir.

- Bunun method tanımı şu şekildedir.

```
int add ( int number1, int number2 ) {  
    int sum = number1 + number2 ;  
    return sum ;  
}
```

Sonuç geriye döner

Method için bunlar kullanılır

Basit Parametreler Kopyalanır.

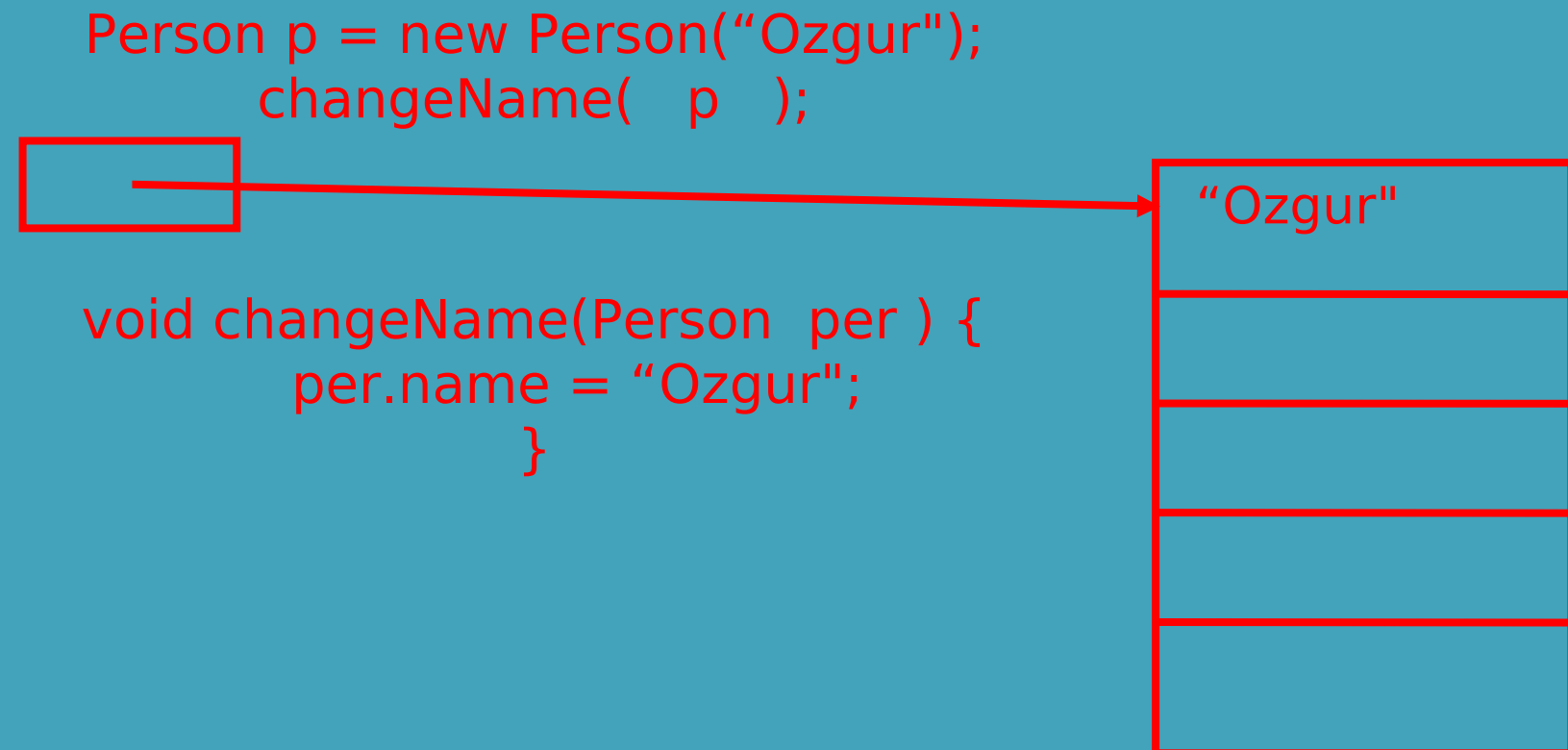
```
int m = 3;  
int n = 5;  
result = add( m , n );
```

```
int add ( int number1 , int number2 ) {  
    while (number1 > 0) {  
        number1--;  
        number2++;  
    }  
    return number2 ;  
}
```

- Buna **değer ile çağırma** (call by value) denir

Nesnelerde ise *referenslar* gönderilir

p , Person in referansını tutar.



- *p* ve *per* aynı nesneyi işaret etmektedir.!
- Bu sebeple *p* de yapılacak değişiklik diğerini de etkileyecektir.
- Buna referans ile çağırma (*call by reference*) denir

NullPointerException

- Bir nesne tanımladığınız zaman onun ilk değeri null (0000) dır.
- null her nesne tanımlaması için kullanılabilen bir değerdir.
 - Bir nesnenin değerinin null olup olmadığını kontrol edebilirsiniz (if (x==null))
a*****)
 - Bir nesneye null değer atayabilirsiniz. (x=null;)
 - null değere sahip bir nesneyi başka şekilde kullanamazsınız.
- Örnek:

Person ozgur;

ozgur.name = "OzgurKoray"; // NullPointerException

Eşitlik

- İki basit değişkenin eşitliğini değerleri üzerinden yapabilirsiniz.

x 24

y 24

- İki nesnenin eşitliğini ise referansları üzerinden yapmanız gerekir.

- $a == b$, fakat $a != c$
- *Stringler için ise equals* metodunu kullanmanız gerekecektir. `s1.equals(s2)`

a

b

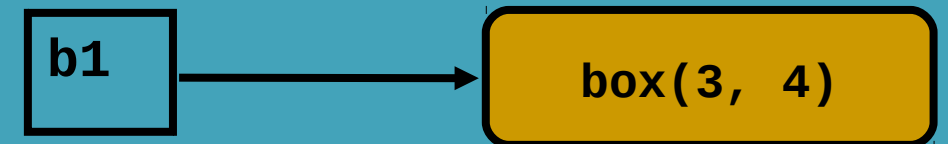
c

OzgurKoray
true
24
5551212

OzgurKoray
true
24
5551212

Nesne Kopyalama

```
Box b1 = new Box(3, 4);
```

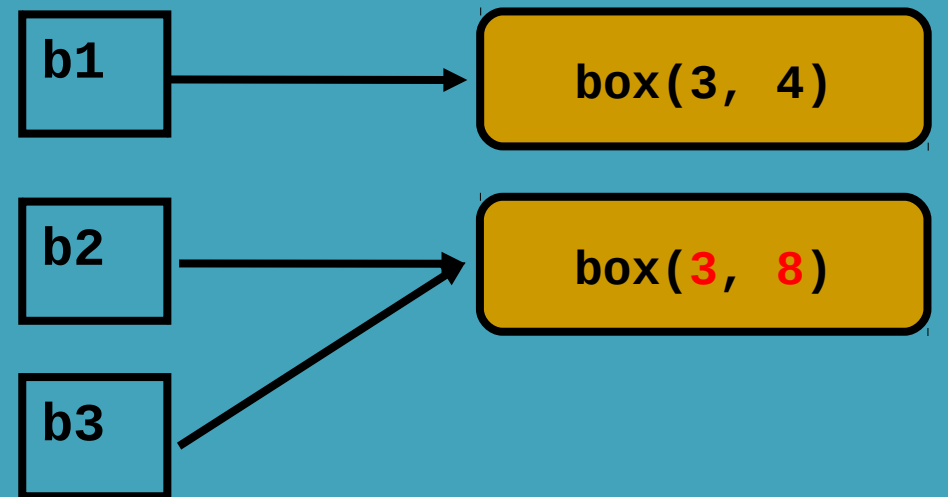


```
Box b2 = b1.clone();
```

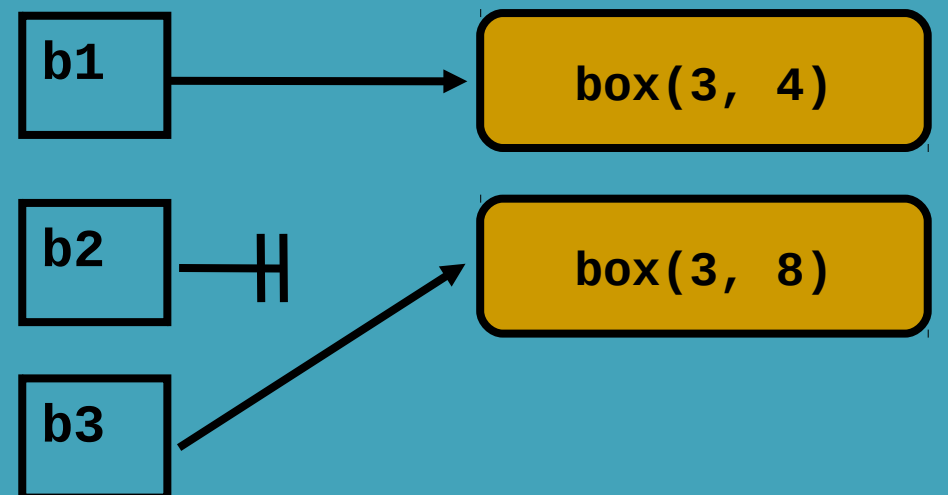
```
Box b3 = b2;
```

```
b2.setWidth(8);
```

```
System.out.println(b3.getWidth());
```



```
b2 = null;
```

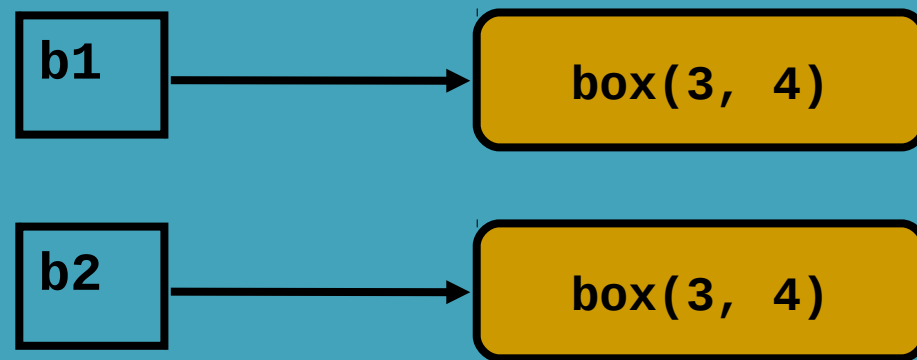


Soru ?

```
Box b1 = new Box(3, 4);  
Box b2 = new Box(3, 4);  
if(b1 == b2)  
    System.out.println("Bu Kutu Nesneleri Eşittir!");  
else  
    System.out.println("Nasıl Eşit Olsunlar ki!.... ");
```


Eşitlik

- Aynı Bellek alanına işaret etmedikleri için eşit değildir.



Eşitlik-String

```
class test {  
    public static void main (String args[]) {  
        String str1= new String("Ozgur");  
        String str2= new String("Ozgur");  
  
        System.out.println ("Ozgur" == "Ozgur");  
        System.out.println (str1 == "Ozgur");  
        System.out.println (str1 == str2);  
        System.out.println (str1.equals(str2));  
        System.out.println (str1.equalsIgnoreCase(str2));  
    }  
} // Çıktı ? ? ?
```

Cıktı

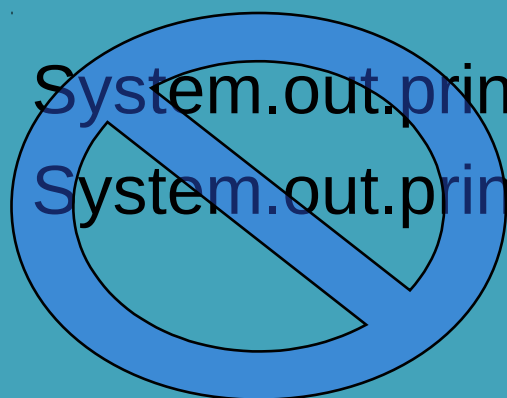
- tue
- false
- false
- true
- true

Adaş Yordamlar-(Metod Overloading)

- Aynı metod ismi vardır. Fakat farklı parametreler gönderilir.
- Örneğin:
 - ❑ bilgiDegistir(String t);
 - ❑ bilgiDegistir(double s);
 - ❑ bilgiDegistir(String t, double s);

Örnek

- `System.out.println(12);`
- `System.out.println("Özgür Koray");`
- `System.out.println('E');`
- `System.out.println(12.0f);`



`System.out.printString("Ozgur Koray");`
`System.out.println(12);`

Adas Yordamlar

```
public class OverLoad {  
    public void same()  
        { System.out.println( "No arguments" ); }  
    public void same( int firstArgument )  
        { System.out.println( "One int arguments" ); }  
    public void same( char firstArgument )  
        { System.out.println( "One char arguments" ); }  
    public int same( int firstArgument ) // Derleme Hatası  
        { System.out.println( "One char arguments" ); return 5; }  
    public void same( char firstArgument, int secondArgument )  
        { System.out.println( "char + int arguments" ); }  
    public void same( int firstArgument, char secondArgument )  
        { System.out.println( "int + char arguments" ); } }  
}
```

CalisanOverloading

```
public class CalisanOverloading {  
    private String isim;  
    private String tel;  
    private double haftalikmaas;  
    public void bilgiYazdir() {  
        System.out.println( isim + "' nin telefonu " + tel + ", Haftalik Maasi " + haftalikmaas);  
    }  
    public void bilgiDegistir(String t) {  
        tel = t;  
    }  
    public void bilgiDegistir(double s) {  
        haftalikmaas = s;  
    }  
    public void bilgiDegistir(String t, double s) {  
        tel = t;    haftalikmaas = s;  
    }  
}  
  
CalisanOverloading ozgur=new CalisanOverloading("Ozgur Koray SAHINGOZ", "6632490", 1234.50);  
ozgur.bilgiDegistir("6632491");  
ozgur.bilgiDegistir(1500.00);  
ozgur.bilgiDegistir("555-3824", 1750.80);
```

Örnek?

Gönderilen integer değerlerin en büyüğünü döndüren Max metodu nasıl geliştirilir.

```
int x= max(12,24);
```

```
int x= max(12,24,35);
```

```
int[] dizi = {12,24,34,45,56,67,78}
```

```
int x=max(dizi);
```


Adaş Yapılandırıcı

- Adaş metodların özel bir durumudur.
- Farklı parametreler ile birden fazla tanımlanabilir.

```
public class CalisanOverCons {  
    CalisanOverCons (String n);  
    CalisanOverCons (String n, String t);  
    CalisanOverCons (String n, String t, double s);  
}
```

Adaş Yapılandırıcı

```
public CalisanOverCons(String n) {  
    isim = n; tel = "000-0000"; haftalikmaas = 0.0;  
}  
public CalisanOverCons(String n, String t) {  
    isim = n; tel = t; haftalikmaas = 0.0;  
}  
public CalisanOverCons(String n, String t, double s) {  
    isim = n; tel = t; haftalikmaas = s;  
}
```

Kullanımı

```
CalisanOverCons ozgur = new CalisanOverCons("Ozgur Koray SAHINGOZ");  
ozgur.chanageinfo("555-1234", 1500.00);
```

```
CalisanOverCons jordan = new CalisanOverCons("Michael Jordan", "555-4548");  
jordan.changeinfo(2500.00);
```

```
CalisanOverCons gates = new CalisanOverCons("Bill Gates", "555-4647", 3000.00);
```

Örnek-2

```
public class Point
{
    public int x = 0;
    public int y = 0;
    public Point(int a, int b)    // yapılandırıcı
    {
        x = a;
        y = b;
    }
    public Point(int a)    // yapılandırıcı
    {
        x = a;
        y = 0;
    }
}
```

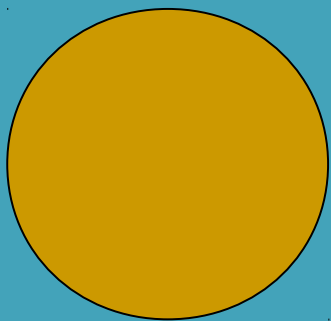
?? Method overloading

Circle Sınıfı ile Kullanım Örneği

```
public class Circle {  
    public double x,y,r;  
    // Constructor  
    public Circle(double centreX, double centreY, double radius)  
    {  
        x = centreX;  
        y = centreY;  
        r = radius;  
    }  
    //Methods to return circumference and area  
    public double circumference() { return 2*3.14*r; }  
    public double area() { return 3.14 * r * r; }  
}
```

Constructor kullanmaz ise

```
Circle aCircle = new Circle();  
aCircle.x = 10.0; // initialize center and radius  
aCircle.y = 20.0  
aCircle.r = 5.0;
```



İlk yaratılma zamanında
çemberin merkezi ve çapı
tanımlı değildir

Bu değerler sonradan atanır.

Çoklu Yapılandırıcılar

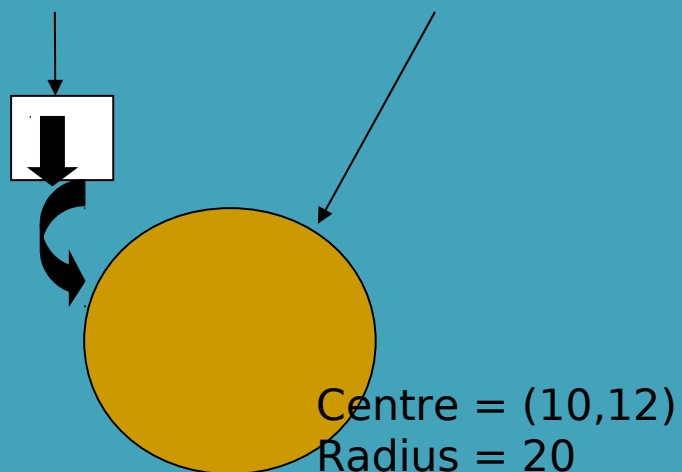
```
public class Circle {  
    public double x,y,r; //instance variables  
    // Constructors  
    public Circle(double centreX, double centreY, double radius) {  
        x = centreX; y = centreY; r = radius;  
    }  
    public Circle(double radius) { x=0; y=0; r = radius; }  
    public Circle() { x=0; y=0; r=1.0; }  
  
    //Methods to return circumference and area  
    public double circumference() { return 2*3.14*r; }  
    public double area() { return 3.14 * r * r; }  
}
```

Kullanım

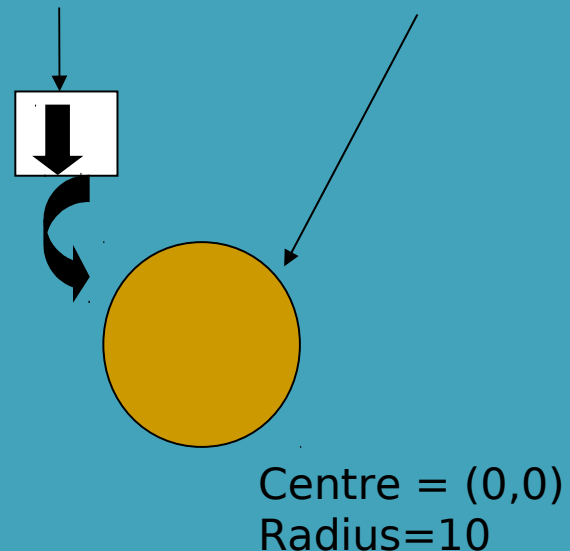
```
public class TestCircles {
```

```
    public static void main(String args[]){  
        Circle circleA = new Circle( 10.0, 12.0, 20.0);  
        Circle circleB = new Circle(10.0);  
        Circle circleC = new Circle();  
    }  
}
```

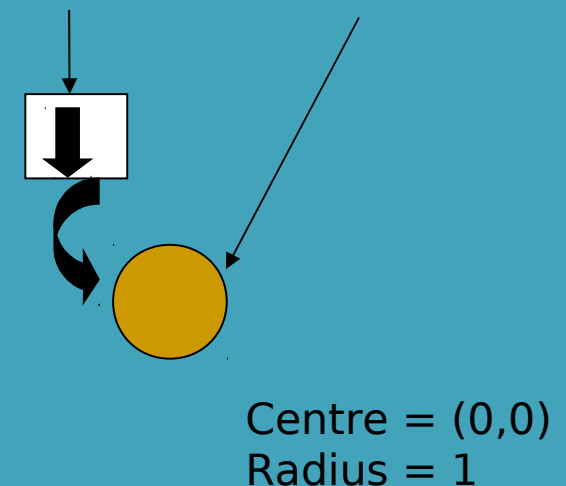
circleA = new Circle(10, 12, 20)



circleB = new Circle(10)



circleC = new Circle()



İlk Değerlerin Atanması

- Uygulamalarında üç tür değişken çeşidi bulunur:
 - yerel (*local*) değişkenler,
 - nesneye ait global alanlar ve
 - son olarak sınıfa ait global alanlar (*statik alanlar*).
- Bu değişkenlerin tipleri temel (*primitive*) veya herhangi bir sınıf tipi olabilir

Değişken Gösterimi

```
public class DegiskenGosterim {  
    int x ;           //nesneye ait global alan  
    static int y ;    // sınıfa ait global alan  
    public void metod () {  
        int i ; //yerel degisken  
    }  
}
```

İlk Değer Alma Sırası

- Nesnelere ait global alanlara başlangıç değerleri hemen verilir; üstelik, yapılandırıcılardan (*constructor*) bile önce...
- Belirtilen alanların konumu hangi sırada ise başlangıç değeri alma sırasında aynı olur.

Örnek

```
class Kagit {  
    public Kagit(int i) {  
        System.out.println("Kagit (" + i + ") ");  
    }  
}
```

Örnek (devam)

```
public class Defter {  
    Kagit k1 = new Kagit(1);           // dikkat  
    public Defter() {  
        System.out.println("Defter() yapilandirici ");  
        k2 = new Kagit(33);  
    } //artık başka bir Kagit nesnesine bağlı  
  
    Kagit k2 = new Kagit(2);           //dikkat  
  
    public void islemTamam() {  
        System.out.println("Islem tamam"); }  
  
    Kagit k3 = new Kagit(3);           //dikkat  
}
```

Çıktı

Kagit (1)

Kagit (2)

Kagit (3)

Defter() yapilandirici

Kagit (33)

Islem tamam

This

```
public class BankAccount {  
    public float balance;  
    public BankAccount( float initialBalance ) {  
        this.balance = initialBalance; } // this şart değil  
    public BankAccount( int balance ) {  
        this.balance = balance; } // this şart  
    public void deposit( float amount ) {  
        balance += amount ; }  
    public String toString() {  
        return "Account Balance: " + balance; }  
}
```

This-2

```
public class Point {  
    public int x = 0;  
    public int y = 0;  
    public Point(int x, int y)    // constructor  
    { this.x = x;  
      this.y = y; }  
    public Point(int x)    // constructor  
    { this.x = x;  
      this.y = 0; }  
}
```


Nesne Olarak “This”

```
public class BankAccount {  
    public float balance;  
    public BankAccount( float initialBalance ) {  
        this.balance = initialBalance;  
    }  
    public BankAccount deposit( float amount ) {  
        balance += amount ;  
        return this;  
    }  
}
```

This

```
public class RunBank {  
    public static void main( String args[] ) {  
        BankAccount richStudent = new BankAccount( 10000F );  
        richStudent.deposit( 100F ).deposit( 200F ).deposit( 300F );  
        System.out.println( "Student: " + richStudent.balance );  
    }  
} //?? Çıktı ne olur
```

Yapılandırıcılarda This

```
public class Tost {  
    int sayi ;  
    String malzeme ;  
  
    public Tost() {  
        this(5);  
        // this(5,"sucuklu");   Hata!-iki this kullanılamaz  
        System.out.println("parametresiz yapılandırıcı");  
    }  
  
    public Tost(int sayi) {  
        this(sayi,"Sucuklu");  
        this.sayi = sayi ;  
        System.out.println("Tost(int sayi) " );  
    }  
}
```

Örnek

```
public Tost(int sayi ,String malzeme) {  
    this.sayi = sayi ;  
    this.malzeme = malzeme ;  
    System.out.println("Tost(int sayi ,String malzeme) " );    }  
  
public void siparisGoster() {  
    // this(5,"Kasarli");      !Hata!-sadece yapılandırıcılarda  
    System.out.println("Tost sayisi="+sayi+ "=" + malzeme );  
}  
  
public static void main(String[] args) {  
    Tost t = new Tost();  
    t.siparisGoster();  
}          // main  
}          // class
```

Yapılandırıcılarda This

- Yapılandırıcılar içerisinde this ifadesi ile her zaman başka bir yapılandırıcı çağrılabilir.
- Yapılandırıcı içerisinde, diğer bir yapılandırıcıyı çağırırken this ifadesi her zaman ilk satırda yazılmalıdır.
- Yapılandırıcılar içerisinde birden fazla this ifadesi ile başka yapılandırıcı çağrılmaz.

Statik Yordamlar (Static Methods)

```
public class StatikTest {  
  
    public static void hesapla(int a , int b) {  
        islemYap(a,b);  
        // !Hata!  
    }  
  
    public void islemYap(int a , int b) {  
        // nesneye ait bir yordam, static bir yordamı çağırabilir  
        hesapla(a,b);  
    }  
}
```

Örnek

```
public class MutluAdam {  
    private String ruh_hali = "Mutluyum" ;  
  
    public void ruhHaliniYansit() {  
        System.out.println( "Ben " + ruh_hali );    }  
  
    public void tokatAt() {  
        if( ruh_hali.equals("Mutluyum" ) )  
            ruh_hali = "Sinirlendim";    }  
  
    public void kucakla() {  
        if( ruh_hali.equals( "Sinirlendim" ) )  
            ruh_hali = "Mutluyum";    }  
}
```

Örnek-2

```
public static void main(String[] args) {  
    MutluAdam obj1 = new MutluAdam();  
    MutluAdam obj2 = new MutluAdam();  
  
    obj1.ruhHaliniYansit();  
    obj2.ruhHaliniYansit();  
  
    obj1.kucakla();  
    obj2.tokatAt();  
  
    obj1.ruhHaliniYansit();  
    obj2.ruhHaliniYansit();  
}  
}
```


Static Metod

```
public class Toplama {  
  
    public static double topla(double a , double b ) {  
        double sonuc = a + b ;  
        return sonuc ;  
    }  
} // Sınıfın durum değişkenlerine bağımlı değil
```

Nasıl Kullanılır

```
public class ToplamIslemi {  
    public static void main(String args[]) {  
  
        String s1=Klavye.stringOku();  
        String s2=Klavye.stringOku();  
  
        double a = Double.parseDouble(s1);  
        double b = Double.parseDouble(s2);  
  
        double sonuc = Toplama.topla(a,b);    // dikkat  
        System.out.println("Sonuc : " + sonuc );  
    }  
}
```

Static Metodlar

- Aynı zamanda sınıf metodları olarakta bilinirler.
- Static metodlar doğrudan sınıfın ismiylede çağrılabilirler.
 - ▣ `double rand = Math.random();`
- Bir static metod, static olmayan bir değere veya metoda erişemez.
 - ▣ Static metodlar nesne örneklerinden bağımsız olarak çalışırlar.

Final Değişkenler

- Sabit değişkenler olup değerlerini değiştiremezsiniz

```
public class CityName {  
    final static public String name = "Kayseri";  
    public static void main(String args[]) {  
        System.out.println("Sehrin Adi " + name );  
    }  
}
```

- CityName.java, sınıfının herhangi bir yerinde *name* değişkeninin değerini değiştiremezsiniz. Çünkü *name* bir final değişkendir.

FINAL

final anahtar sözcüğünün kullanımı ile tanımlanır.

- ❑ `final int INCREMENT = 5;`
- ❑ `INCREMENT` değişkeninin değeri 5 tir
- ❑ final değişkenlere sadece bir kez değer atanabilir.

final int d;

d=5; //olur

d=9; //olmaz

Örnek (Final)

```
class FinalDegisken {  
    public static int x ;  
    public final int y =5;  
    public void ekranaBas(FinalDegisken sd ) {  
        System.out.println(" x = " + x + " y = " + y );  
    }  
public static void main(String args[]) {  
    FinalDegisken sd1 = new FinalDegisken ();  
    sd1.x = 50 ;  
    sd1.y = 2 ;  
    sd1.ekranaBas(sd1);  
    }  
}
```

// Derleme Hatası

Kalıtım

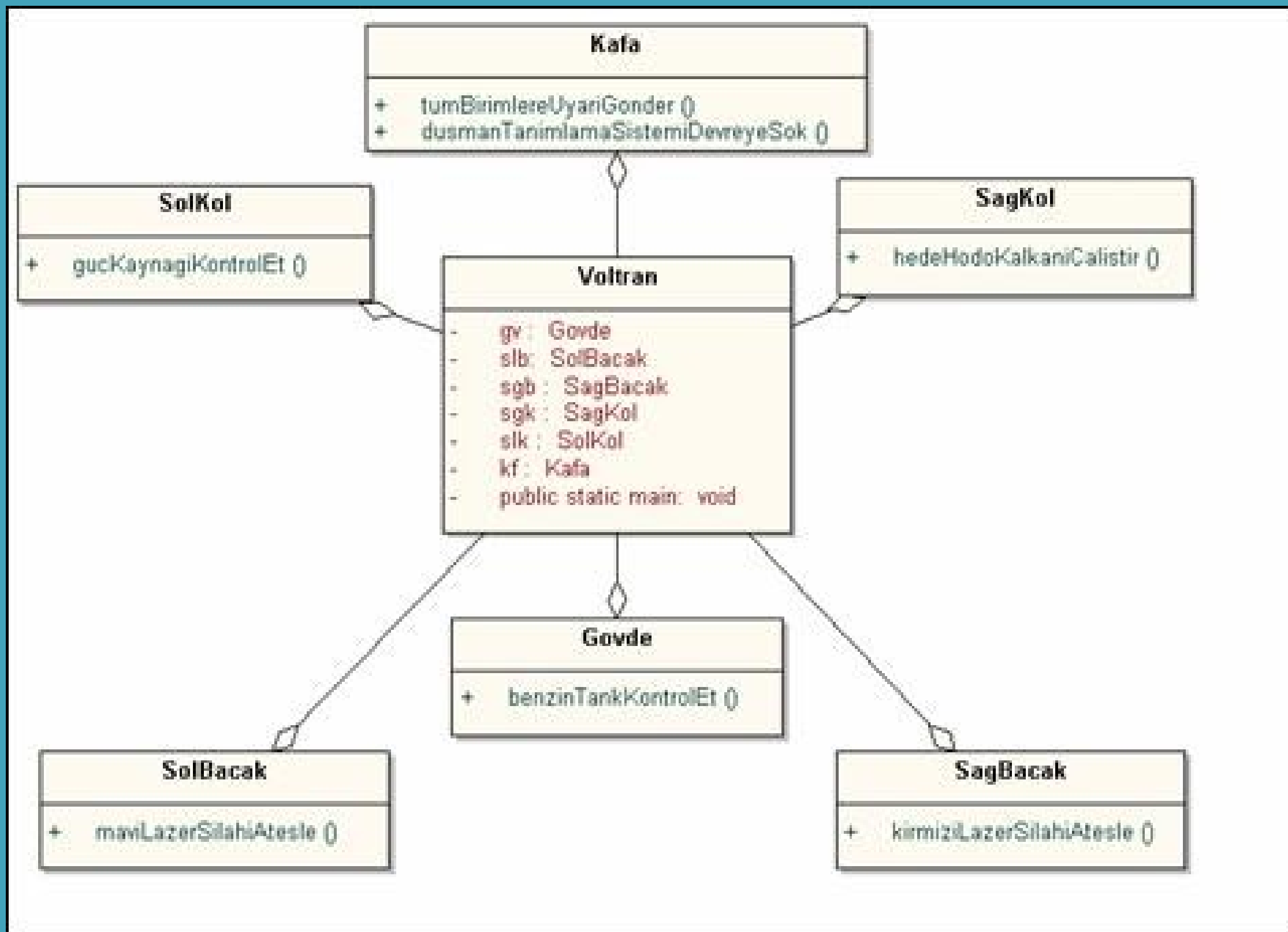
Inheritance

İki Yöntem

Kompozisyon (Composition)

Kalıtım (Inheritance)

Kompozisyon - Voltran



Kalıtım

En basit anlamda, örneğin “Derya, annesinin gözlerini almış” dediğimde, tıp uzmanlarının buna getirdikleri yorum " Derya annesinden kalıtımsal olarak bu özellikleri almış" olur.

Programlama dillerinde de kalıtımın rolü aynıdır.

Zaten nesne yönelimli programlama dillerini tasarlayan uzmanlar, gerçek hayat problemlerini, bilgisayar ortamına taşıyabilmek amacıyla en etkili modelleri geliştirmişler. İşte bu model içerisine kalıtımda katarak çok önemli bir özelliğin kullanılabilmesini sağlamışlar.

En genel tanımı ile kalıtım, **"bir sınıftan yeni sınıflar türetmektir"**

Kalıtımın Faydası

Herşeyden önce kalıtım yolu ile bir sınıftan, yeni sınıflar türetilebilmesinin, türetilen sınıflara etkisi nedir?

- Bu sorunun cevabı kalıtımın da özünü oluşturmaktadır. **Türetilen her bir sınıf, türediği sınıfın özelliklerindeki devralır.**
- Buradan, türetilmiş bir sınıf içerisinde, türediği sınıfa ait üyelere erişilebileceği sonucunu çıkartabiliriz.

Kalıtım

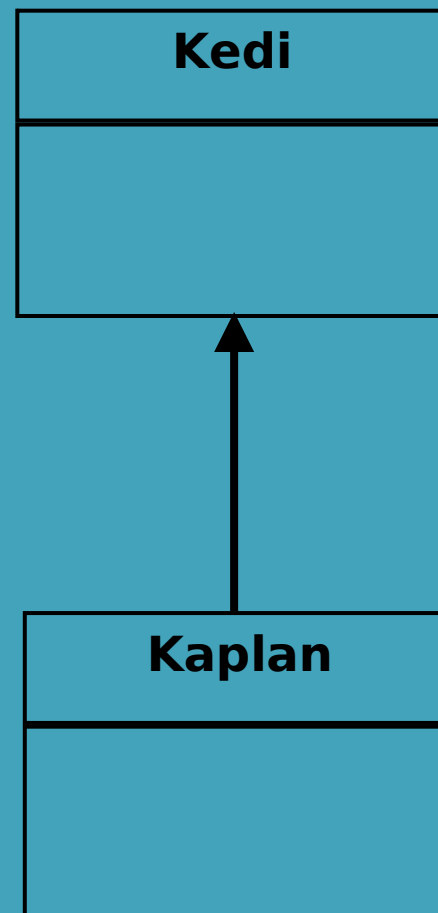
Temel bir sınıftan (*üst sınıf*) birden fazla sınıf türetebilirsiniz.

Ancak bir sınıfı birden fazla sınıftan türetmeniz mümkün değildir.

Bunu yapmak için arayüzler (*interface*) kullanılır.

Türeyen bir sınıf türediği sınıfın tüm özelliklerine ve işlevlerine sahiptir ve türediği sınıftaki bu elemanların yeniden tanımlanmasına gerek yoktur.

Kalıtım - UML



Kalıtım (Inheritance)

```
class Kedi {  
    //..  
}
```

```
class Kaplan extends Kedi {  
    //..  
}
```

KALITIM

Kaplan, **Kedi** den türetilmiştir

Kaplan : alt sınıf (subclass)

Kedi : üst sınıf (superclass)

Kaplan, **Kedi**'nin *public* ve *protected* tüm elemanlarına erişebilir (tam tersi geçerli değil)

TANIMLAMA

```
[Modifier] class ClassName  
  [extends SuperClass]  
  [implements Interface1,  
  Interface2,...]  
  {  
    // Sınıf alanları ve metotları  
  }
```

Türetme işi extends anahtar sözcüğü ile yapılıyor

Örnek

```
class Kedi {
```

```
    protected int ayakSayisi = 4 ;
```

```
    public void yakalaAv() {
```

```
        System.out.println("Kedi sinifi Av yakaladi");
```

```
    }
```

```
    public static void main(String args[]) {
```

```
        Kedi kd= new Kedi() ;
```

```
        kd.yakalaAv() ;
```

```
    }
```

```
}
```

```
class Kaplan extends Kedi {
```

```
    public static void main(String args[] ) {
```

```
        Kaplan kp = new Kaplan();
```

```
        kp.yakalaAv();
```

```
        System.out.println("Ayak Sayisi = " + kp.ayakSayisi);
```

```
    }
```

```
}
```

Örnek

```
class Hayvan {  
    public Hayvan() {  
        System.out.println("Hayvan Yapilandiricisi");    }    }
```

```
class Yarasa extends Hayvan {  
    public Yarasa() {  
        System.out.println("Yarasa Yapilandiricisi");    }    }
```

```
class UcanYarasa extends Yarasa{  
    public UcanYarasa() {  
        System.out.println("UcanYarasa Yapilandiricisi");    }  
    public static void main(String args[]) {  
        UcanYarasa uy = new UcanYarasa();    }    }
```

Çıktı

Hayvan Yapılandırıcısı
Yarasa Yapılandırıcısı
UcanYarasa Yapılandırıcısı

Neden ??

*Hiyerarşideki üst sınıfların boş
yapılandırıcıları çalıştırılır.*

Örnek

```
class Grafik {
    double Taban;    double Yukseklik;
    public Grafik(double a,double b)    {
        Taban=a;    Yukseklik=b;    }
}
class Ucgen extends Grafik {
    String UcgenTuru;
    public Ucgen(double yc,double yuk,String tip)    {
        super(yc,yuk);    UcgenTuru=tip;
    }
    public double Alan() {
        return (Taban*Yukseklik)/2;
    }
}
public class Program {
    public static void main(String[] args)    {
        Ucgen u=new Ucgen(10,20,"Eskenar");
        System.out.println("Ucgen alani "+u.Alan());
    }
}
```

Kompozisyon mu Kalıtım mı?

Hangi Yöntem, Nerelerde Tercih Edilmeli?....

İlişkiler

- "is a" – bir Kalıtım
 - Nesneler başka bir sınıfın altsınıfıdırılar
- "has a" – var Kompozisyon
 - Nesne içinde başka sınıf üyeleri-örnekleri vardır.

Örnek

Employee “is a” BirthDate; //Yanlış!

Employee “has a” Birthdate; //Kompozisyon

Kalıtım

*Sınıflar arasında **bir** ilişkisi olmalıdır*

*UçanYarasa **bir** Yarasadır;*

*Yarasa **bir** Hayvandır;*

*O zaman UçanYarasa'da **bir** Hayvandır.*

*Hayvan'da **bir** Nesnedir.*

İptal Etmek-Ağır Basamak (Overriding)

Ana sınıf içerisinde tanımlanmış bir yordam, ana sınıftan türeyen bir alt sınıfın içerisinde iptal edilebilir.

Ağır Basmak

```
class Kitap {  
    public int sayfaSayisiOgren() {  
        System.out.println("Kitap - sayfaSayisiOgren() ");  
        return 440;  
    }  
    public double fiyatOgren() {  
        System.out.println("Kitap - fiyatOgren() ");  
        return 2500000 ;  
    }  
    public String yazarIsmiOgren() {  
        System.out.println("Kitap - yazarIsmiOgren() ");  
        return "xy";  
    }  
}
```

Ağır Basamak – 3

```
class Roman extends Kitap {  
    public int sayfaSayisiOgren() {  
        System.out.println("Roman - sayfaSayisiOgren() ");  
        return 569;  
    }  
    public double fiyatOgren() {  
        System.out.println("Roman - fiyatOgren() ");  
        return 8500000 ;  
    }  
    public static void main( String args[] ) {  
        Roman r = new Roman();  
        int sayfasayisi = r.sayfaSayisiOgren();  
        double fiyat = r.fiyatOgren();  
        String yazar = r.yazarIsmiOgren();  
    }  
}
```

Student.java

```
public class Student {  
    protected String name;  
    public Student(String name) {  
        this.name = name;  
    }  
    public String toString() {  
        return "Student: " + name;  
    }  
}
```

Super

Altsınıf, üstsınıf'taki metotları “override-ağır basmak” edebilir.

Bu durumda, altsınıf nesnesi, üstsınıfta tanımlı metodu (overriden) nasıl çağırır?

Cevap

`super.method1(...)`

Overriding mi Overloading mi?

Overriding

- Aynı Metod İsmi
- Aynı İmza (Parametre)
- Metotlardan biri ana (ata) sınıfta

Overloading

- Aynı Metod İsmi
- Farklı İmza
- İki metodda aynı sınıfta

Türeyen sınıfta geçersiz hale getirilen metod, temel sınıftaki metodlar ile ya aynı erişim belirleyicisine sahip olmalı yada daha erişilebilir bir erişim belirleyicisi kullanılmalıdır.

Polymorfizm

Çok Formluluk

Giriş

Nesneye Yönelik Programlamanın 3 temel mekanizması vardır. Bunlar:

- Kapsülleme (Encapsulation)
- Kalıtım (Inheritance)
- Çok Formluluk (Polymorphism)

Bunlardan ilk ikisi daha önce gördüğümüz konulardır.

Çok formluluk tek metod ismi ile çok sayıda işlevin yerine getirilebilmesini ifade etmektedir.

Polymorfism (Çok Formluluk)

Bir değişken birden fazla tip değişkene/nesneye işaret edebilir mi?

Farklı tip verileri programın çalışması boyunca saklayabilir mi?

Integer, String, float, double, Öğrenci gibi verileri aynı değişken üzerinden tutabilir miyiz?

Örnek

```
class A {
```

```
    public String toString(){
```

```
        return "AAAAAA";  }
```

```
}
```

```
class B {
```

```
    public String toString(){
```

```
        return "BBBBBB";  }
```

```
}
```

```
class C {
```

```
    public String toString(){
```

```
        return "CCCCCC";  }
```

```
}
```

Örnek

```
public static void main(String[] args) {  
    Object [] array = new Object[3];  
    array[0] = new A();  
    array[1] = new B();  
    array[2] = new C();  
    for (int i = 0; i < 3; i++)  
        System.out.println(array[i]);    // ??  
}
```

`System.out.println(array[i].toString());`

ÇIKTI:

```
AAAAAA  
BBBBBB  
CCCCCC
```

Polymorphic değişken

Bir önceki örnekteki `array[i]` bir polymorphic değişkendir.

Object sınıfından bir değişken olmasına rağmen A, B and C sınıfından nesnelere olan referansları tutmaktadır.

Böyle bir işlem Kalıtım ile ilgilidir.

Polimorfizm

Polimorfizm, nesneye yönelik programlamanın önemli kavramlarından biridir ve sözlük anlamı olarak "bir çok şekil" anlamına gelmektedir.

Polimorfizm ile kalıtım konusu iç içedir. A,B ve C sınıfları Object sınıfının bir alt sınıfı olduğu için bu verileri *array[i]* içinde saklayabildik.

Örnek

```
class Asker {  
}
```

```
class Er extends Asker {  
public void selamVer() {  
    System.out.println("Er Selam verdi");  
}  
}
```

```
class Yuzbasi extends Asker {  
public void selamVer() {  
    System.out.println("Yuzbasi Selam verdi");  
}  
}
```

Örnek

```
public class PolimorfizmOrnekBir {  
public static void hazirOl(Asker a) { // Yazabilirmiyiz ??  
    a.selamVer();  
}
```

```
public static void main(String args[]) {  
    Asker a = new Asker();  
    Er e = new Er();  
    Yuzbasi y = new Yuzbasi();  
    hazirOl(a); // Yazabilirmiyiz ??  
    hazirOl(e);  
    hazirOl(y);  
} }
```

Cevap

Bu olay polimorfizmden kaynaklanmaktadır.
Hangi metodun çağrılacağıнын belirlenmesi
polimorfizmi ifade etmektedir.

Hangi metodun çağrılacağı derleme anında değil
çalışma anında bellidir.

Yani ilgili metodun bağlantısı sonradan (geç)
yapılmaktadır. Bu nedenle bu bağlantıya **Geç**
Bağlama denir.

Geç Bağlama ile *Polimorfizm* içiçedir. Eştir.

Geç Bağlama

```
Asker a=new Er();  
a.selamVer();
```

Bu size ilk başta hata olarak gelebilir.

Ama arada kalıtım ilişkisinden dolayı (Er **bir** Askerdir) nesneye yönelik programlama çerçevesinde bu olay doğrudur.

Şimdi, hangi nesnesin **selamVer()** yordamı çağrılacak

- *Asker* nesnesinin mi?
- Yoksa *Er* nesnesinin mi ?

Cevap

Er nesnesinin `selamVer()` yordamı çağrılacaktır.
Çünkü *Asker* tipindeki yerel değişken (a) *Er nesnesine*
bağlanmıştır.

Eğer *Er nesnesinin* `selamVer()` yordamı olmasaydı o zaman *Asker nesnesine* ait olan `selamVer()` yordamı çağrılacaktı.

Fakat *Er sınıfının* içerisinde, ana sınıfa ait olan (*Asker sınıfı*) `selamVer()` yordamı iptal edildiğinden (*override*) dolayı, *Er nesnesinin* `selamVer()` yordamını çağırılacaktır.

Geç Bağlama

Peki hangi nesnesinin selamVer()
yordamının çağrılacağı ne zaman belli
olur?

- Derleme anında mı (*compile-time*)?
- Yoksa çalışma anında mı (*run-time*)?

Geç Bağlama

Cevap; çalışma anında (*run-time*).
Bunun sebebi, derleme anında `hasIR()`
yordamına hangi tür nesneye ait
referansın gönderileceğinin belli
olmamasıdır.

Geç Bağlama

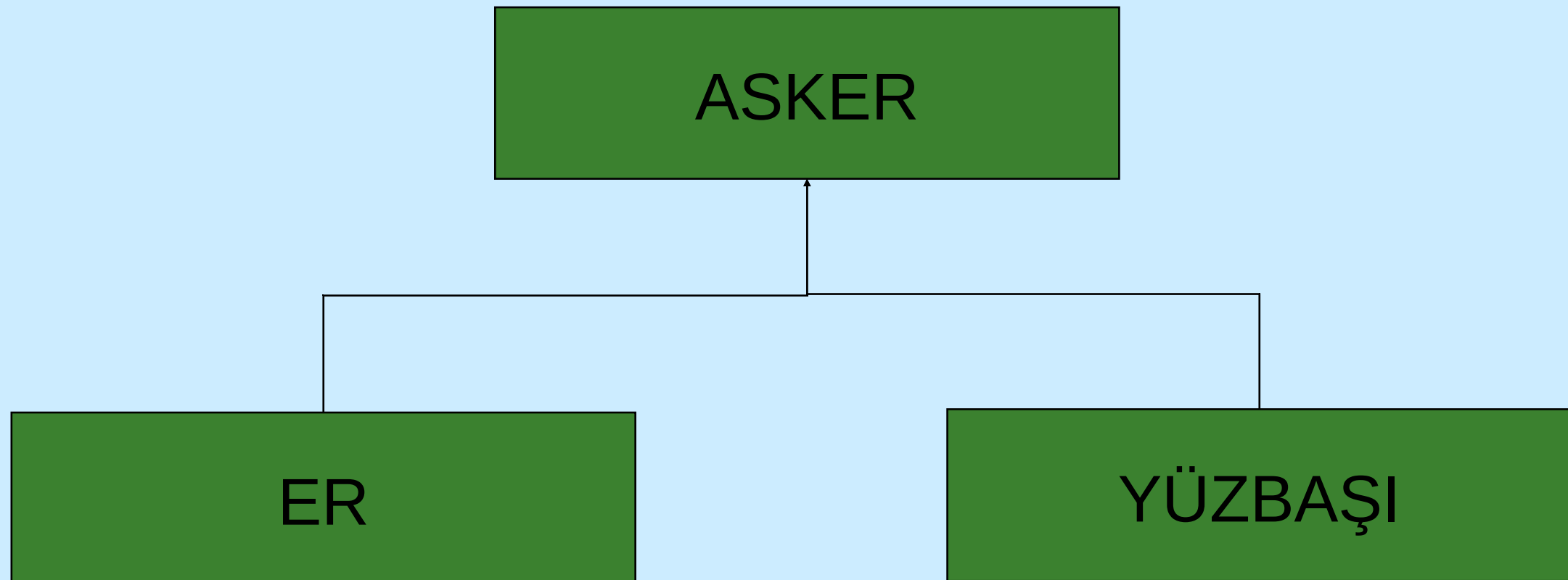
Derleme anında (*compile-time*) hangi nesneye ait yordamın çağrılacağını **bilinemiyorsa** buna geç bağlama denir.

Geç bağlamanın diğer isimleri

- Dinamik bağlama (*Dynamic binding*)
- Çalışma anında bağlama (*Run-time binding*)

Bunun tam tersi ise *erken bağlamadır* (*early binding*).

Hiyerarşik Yapı



Yukarı Çevirim (*upcasting*)

Yukarı Çevrim Kalıtım ile ilgili bir konudur.

- Er **bir** Askerdir, veya
- Yüzbaşı **bir** Askerdir, diyebiliriz.

Yani **Asker** sınıfının yaptığı her işi Er sınıfı veya **Yuzbasi** sınıfı da yapabilir + türetilen bu iki sınıf kendisine has özellikler taşıyabilir.

Asker sınıfı ile **Er** ve **Yuzbasi** sınıflarının arasında kalıtımsal bir ilişki bulunmasından dolayı, **Asker** tipinde parametre kabul eden `hazirOl()` yordamına **Er** ve **Yuzbasi** tipindeki referansları paslayabildik, bu özelliğinde *yukarı çevirim* denir

Tanımlama ve Yaratma

Asker a = **new** Asker() ;

Asker a = **new** Er();

Asker a = **new** Yuzbasi();

Aşağıya Çevirim(Down Casting)

Aşağıya çevirim tehlikelidir.

Daha **genel** bir tipden daha **özellikli** bir tipe doğru çevirim vardır.

Yanlış bir çevirim yapıldığında, çalışma anından (*run-time*) istisna (*exception*) fırlatılır/verilir.

Aşağıya Çevirim

1. `Er e = new Er();`
2. `Asker a = e; // Yukarı Çevrim`
3. `e = (Er) a; // Aşağı Çevrim`
4. `Yuzbasi y = (Yuzbasi) a; // Aşağı Çevrim`
5. `Ogrenci o = (Ogrenci) a; // Derleme Hatası`
`// inconvertible types`

instanceof

Kalıtımla alakalı bir operatördür.

Bir nesnenin bir sınıfın örneği olup olmadığını kontrol etmekte kullanılır.

```
Object o = new Integer(2);
```

```
System.out.println(o instanceof Object);
```

```
System.out.println(o instanceof Integer);
```

```
System.out.println(o instanceof Ogrenci);
```

final ve Geç bağlama

Final anahtar kelimesinin değişkenlerde kullanılması **sabitlik** ifade etmektedir. (sabit değişkenler)

Final özelliğinin kullanılmasının iki sebebi olabilir.

- Tasarım
- Verimlilik

Örnek

```
class Kedi2 {  
    public final void yakalaAv() {  
        System.out.println("Kedi sinifi Av yakaladi");  
    }  
}  
class Kaplan2 extends Kedi2 {  
    public static void goster(Kedi2 k) {  
        k.yakalaAv();  
    }  
    public void yakalaAv() {                // iptal edilemez  
        System.out.println("Kaplan sinifi Av yakaladi");  
    }  
    public static void main(String args[] ) {  
        Kedi2 k = new Kedi2() ;  
        Kaplan2 kp = new Kaplan2();  
        goster(k);  
        goster(kp);  
    }  
}
```

Soyut Sınıflar
Abstract Classes

Somut Sınıflar

Şimdiye kadar görmüş olduğumuz sınıflar *örneklenebilir*(*can be instantiated*) sınıflar idi.

```
Node node = new Node(12f);
```

Yani “new” anahtar kelimesi ile o sınıfın örneği olan bir nesne yaratabiliyorduk.

Bu şekilde örneklenebilen(yaratılabilen) sınıflara *somut* (*concrete*) *sınıf* adı verilir.

Soyut Sınıflar

Bazı durumlarda ise doğrudan örneklenemeyen sınıf yapılarının geliştirilmesi faydalı olabilir.

Bu tip sınıflar *soyut (abstract) sınıf* olarak adlandırılırlar.

Bu sınıfların amacı alt sınıfların fonksiyonelliklerini belirlemektir.

Alt Sınıfı Zorlama

Eğer tüm alt sınıfların belirli yeteneklerinin (metodlarının) olmasını istiyorsak ne yaparız!

Örneğin tanımlanan tüm Şekillerin alanlarının ve çevrelerinin hesaplanmasına olanak sağlayan metod çağrılarını geliştirmelerini nasıl zorunlu tutarız ?

Çözüm

Bunun için üst sınıfı geliştiren bizim herhangi bir kod yazmamamız gereklidir.

Kodu alt sınıfı geliştiren kullanıcılara yazdırmamız gerekmektedir.

Ama Nasıl ???

Soyut Metodlar-Boş Metodlar

Soyut sınıflarda herhangi bir kodlama olmasa da bu metodlar boş geliştirilmiş olan metodlardan farklıdır.

Soyut Metodlar-Boş Metodlar

Boş metodlardan sonra metodun başladığını gösteren parantez işaretlerinin kullanılması gereklidir.

Soyut metodlarda ise parantezler kullanılmaz.

- `public void bosMethod() { }`
- `abstract public void soyutMethod();`

Kullanım

Soyut Sınıflardan Nesneler Yaratılamaz

```
// Böyle bir komut derlenmez  
Sekil sekil = new Sekil();
```



Eğer bir alt sınıf geliştirildi ise bütün **soyut (abstract)** metodlar geliştirilmelidir-kodlanmalıdır.

Soyut Sınıflar-2

Eğer alt sınıf bütün soyut metodları gerçekleştirmez ise o zaman bu sınıfta *abstract/soyut* olarak tanımlanması gerekmektedir.

Eğer gerçekleştirilmiş ise o zaman *concrete/somut* sınıf olarak kullanılabilir.

Soyut Sınıfların Özellikleri

Soyut bir sınıftan türetilmiş alt sınıflara ait nesneler, çok rahat bir şekilde yine bu soyut sınıf tipindeki referanslara bağlanabilirler (yukarı çevirim).

Böylece polimorfizm ve geç bağlamanın kullanılması mümkün olur.

Soyut Sınıflar

Bir sınıf, bir veya daha fazla soyut metod içeriyor ise soyut olarak tanımlanmak zorundadır.

Soyut sınıfın soyut metodunun mutlaka alt sınıfta gerçekleştirilmesi gerekmektedir.

Soyut bir yapılandırıcı veya soyut bir static metod tanımlayamayız.

Örnek

```
public abstract class Sekil {  
    protected double x;  
    protected double y;  
  
    public void yerlestir(double xval, double yval) {  
        x = xval;  
        y = yval;  
    }  
  
    public abstract double çevre ();  
  
    public abstract double alan();  
}
```

Dikdortgen Sınıfı

```
public class Dikdortgen extends Sekil {  
    private double genislik;  
    private double yukseklik;  
    public Dikdortgen(double w, double h) {  
        genislik=w;  
        yukseklik=h;    }  
    public void çevre() {  
        return 2*(genislik+yukseklik);    }  
  
    public double alan() {  
        return genislik*yukseklik;    }  
}
```

Örnek

```
abstract class SoyutTemel {  
    public static void Yaz()    {  
        System.out.println("Ben soyut temel sinifim");  
    }  
  
    abstract public void Kimlik();  
  
    SoyutTemel()    {  
        System.out.println("Soyut metoddan once");  
        Kimlik(); // dikkat  
        System.out.println("Soyut metoddan sonra");  
    }  
    int deger=40;  
}
```

Soyutlamada Dikkat Edilen Kurallar

- 1 Soyut sınıflardan nesne örnekleri oluşturulamaz.
- 2 Soyut metod bildirimine sahip olan bir sınıf sadece soyut sınıf olarak tanımlanabilir.
- 3 Soyut sınıflardan türetilen somut sınıflar, soyut metodları mutlaka override etmelidirler.
- 4 Soyut sınıflardan türeyen nesne örneklerini, soyut sınıf nesnelere referans olarak aktararak çok biçimliliğin uygulanmasını sağlayabiliriz.
- 5 Soyut sınıflardan başka soyut sınıflarda türetilebilir.
- 6 Soyut metodları private olarak tanımlayamayız.
- 7 Soyut sınıflara ait yapıcılar tanımlayabiliriz. Ancak bu yapıcıları soyut tanımlayamayız.
- 8 Static bir soyut metod bildirimi yapamayız.
- 9 Soyut sınıflar içerisinde soyut metod bildirimleri dışında, normal metodlar ve alanlar tanımlayabiliriz.

ARRAYÜZLER

Interfaces

```
class Comparable {  
    public boolean compareTo(Object o) { // Niye ??  
        // Kod Var mı? Yok mu?  
        // Soyut Sınıf olur mu??  
    }  
}  
  
class Person extends Comparable {  
    public boolean compareTo(Object o) {  
        if (o instanceof Person) {  
            ???????  
        }  
    }  
}
```

Dezavantajı

Eğer bu sınıfın başka bir sınıfın altsınıfı olması gerekiyor ise olmaz?

extends edince üst sınıftan bazı değerleri kalıtsal olarak alması gerekir

Arayüzler

Kalıtsal olan bir şeye ulaşma ihtiyacımız yoksa extends ile kalıtmaya ihtiyaç yoktur.

Benim ihtiyacım olan tek şey alt sınıfın bir metod ismini geliştirmesini zorunluluk olarak tutmamdır.

O zaman yeni bir yapıya ihtiyacım var.

Comparable Arayüzü

Geliştirilen sınıfın **Comparable** arayüzünü *implement* etmesi sağlanarak bu sınıftan oluşturulan nesnelerin sort metodu içinde sıralanması sağlanabilmektedir.

Comparable arayüzünü implement etmek *compareTo* metodunun geliştirilmesini zorunlu kılmaktadır.

Arayüzler

Arayüzler, soyut (abstract) sınıfların bir üst modeli gibi düşünülebilir.

Soyut sınıfların içerisinde hem iş yapan hem de hiçbir iş yapmayan sadece ***birleştirici*** rol üstlenen gövdesiz yordamlar bulunur.

Bu birleştirici rol oynayan yordamlar, soyut sınıftan (*abstract class*) türetilmiş alt sınıfların içerisinde iptal edilmeleri (*override*) gerekir.

Arayüzlerin içerisinde ise iş yapan herhangi bir yordam bulunamaz; arayüzün içerisinde tamamen gövdesiz yordamlar bulunur.

Arayüz (Interface) ve Soyut Sınıflar

Eğer bir sınıf (soyut sınıflar dahil) bir arayüze (*interface*) ulaşmak istiyorsa, bunu *implements* anahtar kelimesi ile gerçekleştirir.

Ayrıca eğer bir sınıf bir kere arayüze ulaştığı zaman artık onun tüm gövdesiz yordamlarını iptal etmesi (override) gerekir.

Kalıtım mı arayüzler mi?

Arayüzler kalıttan daha farklıdır. Bir üst sınıftan durum değişkenleri veya metodlar kalıtılmaz.

Sadece tanımlı olan metodların isimleri alınarak bunları geliştirilmesi derleyici tarafından zorlanır.

Bu zorlama sınıflar arasında bir ***standart*** oluşturulur.

Arayüzler ve Sınıflar (Karşılaştırma)

Sınıfın rollerini belirler, sınıfın sorumluluklarını tanımlar

Bir nesnenin özelliklerini ve kapasitelerini belirler

Farklı nesnelerin ortak kapasitelerini belirler

Benzer nesnelerin ortak özelliklerini belirler

Metodları tanımlar fakat gerçekleştirmez

Metodları tanımlar ve gerçekleştirir.

Bir sınıf çok sayıda arayüzü implement edebilir.

Bir sınıf sadece bir üst sınıftan türetilebilir.

Syntax

```
interface interfaceName {  
    /* Metod Tanımlamaları */  
}
```

```
class className implements  
    interfaceName {  
    /* Arayü metodlarının geliştirilmesi */  
    /* Sınıfın kendi verileri ve metodları. */  
}
```

Örnek: HareketliNesne

```
interface HareketliNesne {  
    boolean calistir();  
    void      durdur();  
    boolean donus(int derece);  
    double  kalanYakit();  
    boolean hizDegistir(double kmSaat);  
}
```

Ucak

```
class Ucak implements HareketliNesne {  
    boolean calistir() {  
        } //Gerekli Kodlama.  
    void durdur() {  
        } //Gerekli Kodlama.  
    boolean donus(int derece) {  
        } //Gerekli Kodlama.  
    double kalanYakit() {  
        } //Gerekli Kodlama.  
    boolean hizDegistir(double kmSaat) {  
        } //Gerekli Kodlama.  
}
```

Genel hatalar

Bir arayüzden değişken tanımlayabilirsiniz

```
HareketliNesne x;
```

Fakat bir Nesne örnekleyemezsiniz!!!!

```
HareketliNesne x = new HareketliNesne (); // HATA
```

Bu arayüzü gerçekleştirmiş sınıftan bir nesne oluşturup, arayüz değişkenine atayabiliriz.

```
HareketliNesne x = new Ucak(); // OK
```


Arayüzün İçerisinde Alan Tanımlama

Arayüzlerin içerisinde gövdesiz (soyut) yordamların dışında alanlarda bulunabilir. Bu alanlar uygulamalarda **sabit** olarak kullanılabilir.

Çünkü arayüzün içerisinde tanımlanan bir alan (ilkel tipte veya sınıf tipinde olsun) otomatik olarak hem **public** erişim belirleyicisine hem de final ve **static** özelliğine sahip olur.

Örnek

```
interface Aylar {  
    int OCAK    = 1,  
        SUBAT  = 2,  
        MART   = 3,  
        NISAN  = 4,  
        MAYIS  = 5,  
        HAZIRAN = 6,  
        TEMMUZ = 7,  
        AGUSTOS = 8,  
        EYLUL  = 9,  
        EKIM   = 10,  
        KASIM  = 11,  
        ARALIK = 12;  
}
```

ÖRNEK

```
interface Hayvan{  
    public void avlan() ;  
  
}
```

```
abstract class Kedi implements Hayvan{  
  
}
```

Bu Kod Derlenir mi?

Cevap

Derlenir. Aşağıdaki örnekte görüldüğü gibi Arayüzün metodları altsınıfta override (ağır basmak) edilmelidir.

```
class Kaplan extends Kedi {  
    public void avlan() {  
        //*****  
    }  
}
```

Arayüzlerin Kalıtım ile Genişletilmesi

Bir arayüz başka bir arayüzden türetilebilir.

Böylece arayüzler kalıtım yoluyla genişletilmiş olur.

Aynı imzalı metodlar

Arayüzlerin içerisinde dönüş tipleri haricinde herşeyleri aynı olan gövdesiz (soyut) yordamlar varsa bu durum beklenmedik sorunlara yol açabilir.

Sorular ?



Teşekkürler

www.tart.com.tr / cihangir.savas@tart.com.tr

Web

cihangirsavas.com.tr

GitHub

github.com/siesta

Linkedin

tr.linkedin.com/in/cihangirsavas