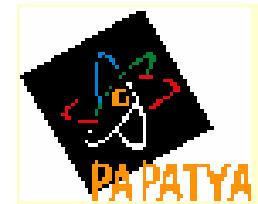


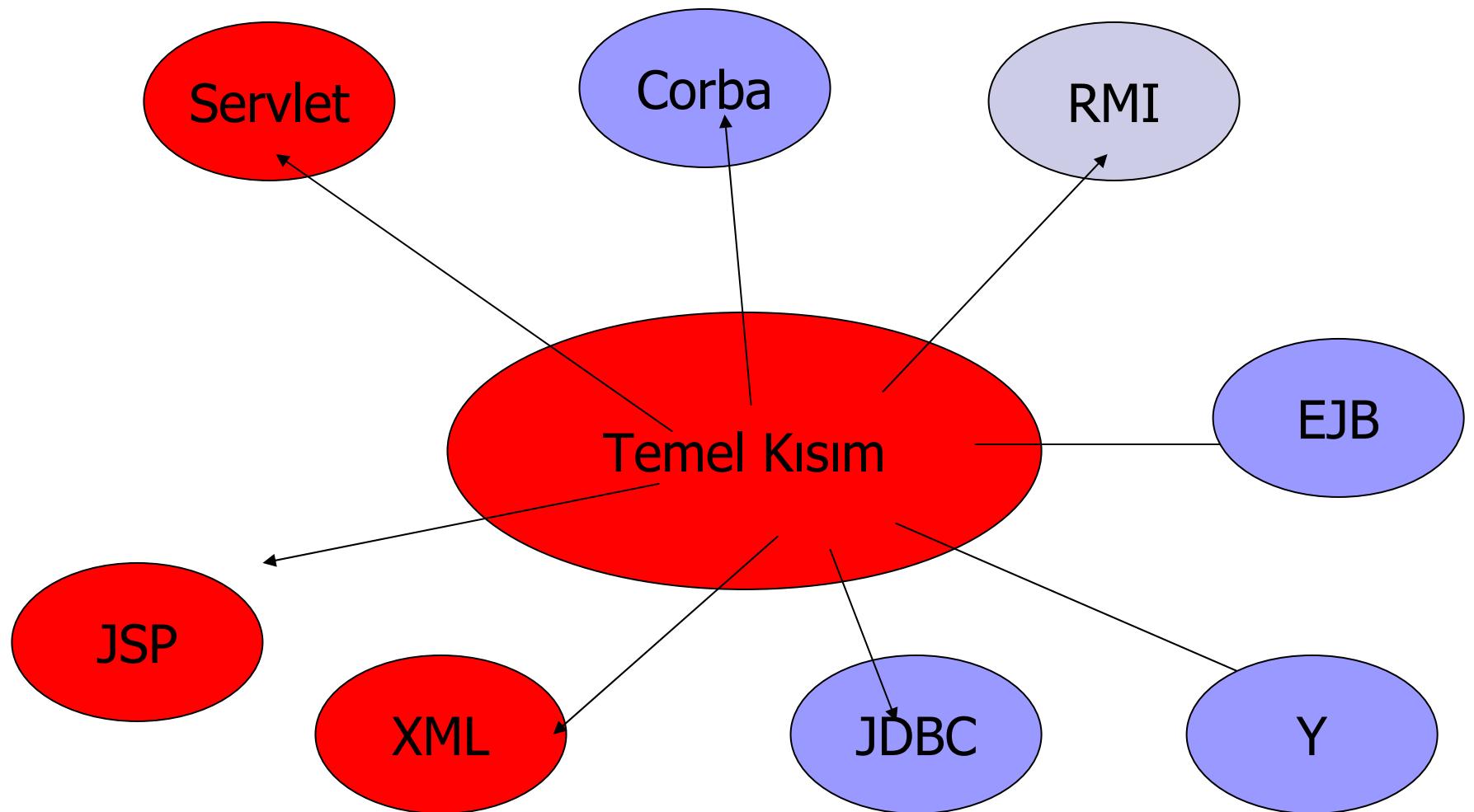
Java

ile

Nesneye Yönelik Programlama



Bu Dönem Hakkında



Java Nedir?

- Java ™ platformu , ağ(network) ‘ın önemi hesaba katılarak ve aynı yazılımın birçok değişik bilgisayar ortamında veya değişik tür makinalarda çalışması fikri ile geliştirilmiş yeni bir teknolojidir.
- Java teknolojisi kullanılarak aynı uygulamayı değişik ortamlarda çalıştırabiliriz – örneğin Pc’lerde , Macintosh bilgisayarlarda, hatta cep telefonlarında.
- Java diğer programlama dilleri gibi başlı başına bir ürün değildir.
- Java ve Java’ya bağlı alt teknolojiler, Sun Microsystems tarafından verilmiş belirtimlerden (specifications) oluşmaktadır.Eğer bu belirtimlere sadık kalınmaz ise hukuki olarak suç işlenmiş olur.

Java İle Neler Yapılabilir?

Java Programlama dili ile projelerimizi diğer programlama dillerine göre daha kolay ve sağlıklı bir şekilde yapmamız mümkündür . Kısaca göz atacak olursak , Java ile ;

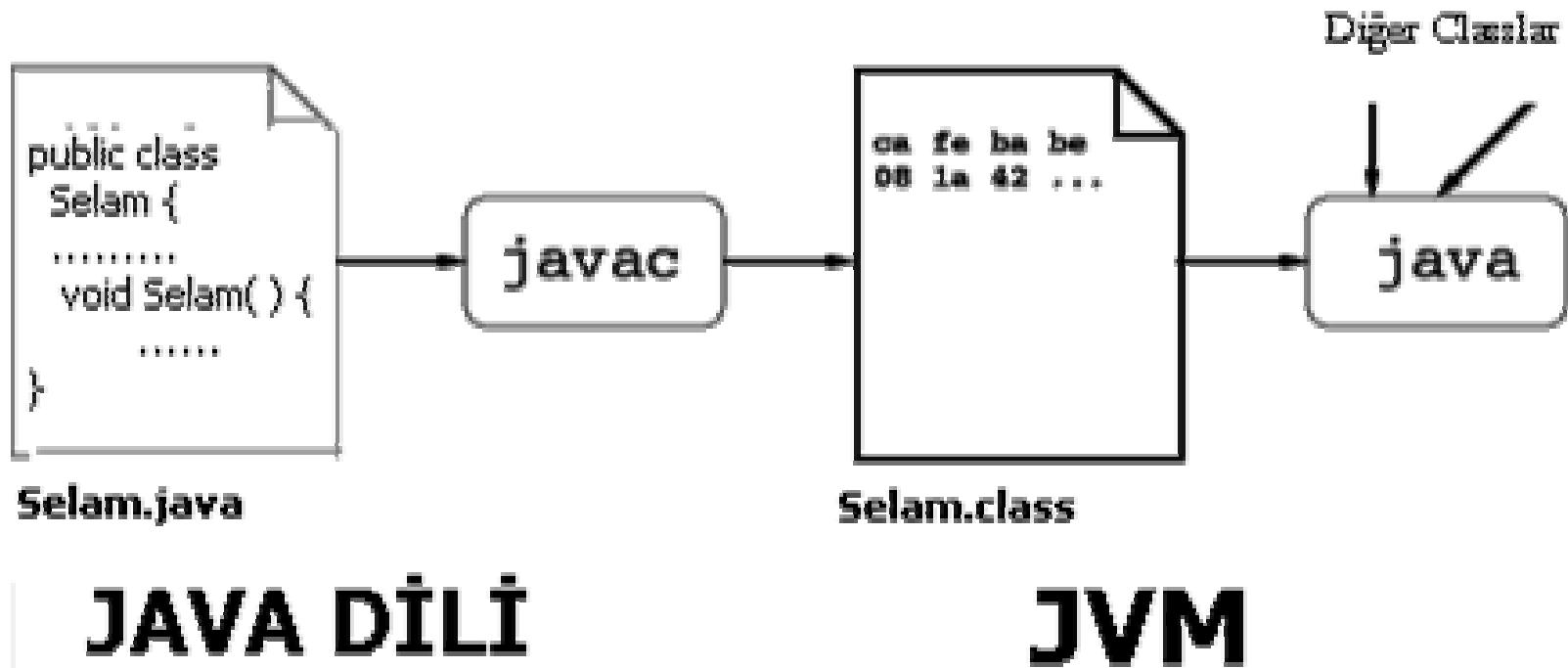
- GUI (graphical user interface , grafiksel kullanıcı ara yüzü) uygulamaları , Appletler.
- Distributed components (ör . EJB, RMI, CORBA).
- Servlet, Jsp (web tabanlı uygulamalar).
- Veri tabanlarına erişim ile alakalı uygulamalar.
- Cep telefonları, Smart kartlar için uygulamalar .
- Ve daha niceleri... için uygulamalar yazmamız mümkündür.

Bir Kere Yaz Her Yerde Çalıştır

- Java uygulamaları JVM (Java Virtual Machine) tarafından yorumlanır(interpreted).
- JVM , işletim sisteminin en tepesinde bulunur
- Java uygulamaları değişik işletim sistemlerinde , herhangi bir değişiklik yapmadan çalışabilir, Java'nın felsefesi olan “bir kere yaz heryerde çalıştır” sözü gerçekleştirılmıştır.



Çalışma Evreleri



Çalışma Evreleri

Derleme anı (Compile time)



Çalıştırma anı (Run time)



Kategoriler

- Java platformunun ana grupları.
 - Standart Java
 - Enterprise Java
 - Gömülü cihazlar için Java (embedded devices)
 - XML Teknolojileri
 - Diğer Teknolojiler

Standart Java

- Java 2 SDK (J2SE)
- Java 2 Runtime Environment
- Java Plug-in
- Java Web Start
- Java HotSpot Server Virtual Machine
- Collections Framework
- Java Foundation Classes (JFC)
- Swing Components
- Pluggable Look & Feel
- Accessibility
- Drag and Drop
- Security
- Java IDL
- JDBC
- JavaBeans
- Remote Method Invocation (RMI)
- Java 2D

Enterprise Java

- J2EE (Java 2 Enterprise Edition)
- CORBA Teknolojisi
- ECperf Teknolojisi
- Enterprise JavaBeans Teknolojisi
- Kontaynerler için Java Yetkilendirme Kontratı (Java Authorization Contract for Containers) (Java ACC)
- Java IDL
- JavaMail API
- Java Mesajlaşma Servisi (Message Service) (JMS) API
- JavaServer Faces
- JavaServer Pages
- Java Servlets
- JDBC Teknolojisi
- J2EE Bağlayıcı Mimarisi (Connector Architecture)
- Hareketler (Transactions)

Gömülü Cihazlar İçin Java (*Embedded Devices*)

- Java 2 Platform, Micro Edition (J2ME technology)
- Java 2 Platform, Micro Edition (J2ME Teknolojisi)
- Bağlı Aygit Konfigurasyonu (Connected Device Configuration) (CDC)
- Sınırlı Bağlanmış Aygit Konfigurasyonu (Connected Limited Device Configuration) (CLDC)
- C Sanal Makinası (Virtual Machine) (CVM)
- K Sanal Makinası (Virtual Machine) (KVM)
- PersonalJava
- Java Card
- JavaPhone API
- Java TV API
- Jini Network Technology
- Mobil Bilgi Aygit Profili (Mobile Information Device Profile) (MIDP)

XML Teknolojileri

- XML İlişkilendirilmesi için Java Mimarisi (Java Architecture for XML Binding) (**JAXB**)
- XML-Tabanlı RPC için JAVA API'si (Java API for XML-Based RPC) (**JAX-RPC**)
- XML Mesajlaşması için JAVA API'si (Java API for XML Messaging) (**JAXM**)
- XML İşlemleri için JAVA API'si (Java API for XML Processing) (**JAXP**)
- XML Kayıtları için JAVA API'si (Java API for XML Registries) (**JAXR**)

Diğer Teknolojiler

- Araç Ürünler
 - MIF Doclet
 - Sun ONE Stüdyo (Studio)
- Ağ (NetWork) Ürünleri
 - Sertifikalı JAIN API Ürünleri (JAIN API Certified Products)
 - Java Dynamic Management Kit
 - Java Yönetim Uzantısı (Java Management Extensions) (JMX)
 - Java MetaData Arabirimi (Java Metadata Interface) (JMI)
 - Java Paylaşılan Veri Araç Takımı Java Shared Data Toolkit
 - Java Spaces Teknolojisi
 - Servis Sağlayıcıları için Java Teknolojisi (Java Technology for Service Providers)
 - Jini Ağ Teknolojisi (Network Technology)
 - JXTA Projesi
 - J2ME Platformu için JXTA Projesi (Project JXTA for J2ME Platform)
 - Sun Chili!Soft ASP

Java'nın Gelişim Evreleri

1995	Java teknolojisinin ilk çıkış yılı ; ilk olarak Applet teknolojisinin dikkat çektiği seneler.
1996	Java Development Kit (JDK) v1.0 çıktı. Temel seviyeli işlevleri içeren bir versiyon (ör. soket programlama, Girdi/Cıktı (Input/Output), GUI (Graphical User Interface- Grafik Kullanıcı Arabirimi)
1997	JDK 1.1 çıktı. Bu sürümde Java GUI , veritabanı erişimi için JDBC , dağınık nesneler için RMI ve daha birçok yeni gelişmeler eklendi .
1998	JDK 1.2 çıktı . JFC/Swing yayınladı- aynı sene içersinde http://java.sun.com internet adresinden 500,000+ adet indirme(download) gerçekleştirildi.
1999	Java teknolojisi J2SE, J2EE ve J2ME olarak 3'e bölündü . Java HotSpot (performans artırmacı) yayınladı . JavaServer Pages (JSP) teknolojisi yayınladı. J2EE platform'u yayınladı . Linux üzerinde J2SE platformu yayınladı .
2000	JDK v1.3 çıktı . Java APIs for XML teknolojisi yayınladı .
2002	JDK v1.4 versiyonu çıkarıldı (Merlin projesi). Java API for XML binding yayınladı.
2003	2003 yılının sonuna doğru JDK v1.5 versiyonun yapılması planlanmaktadır (Tiger projesi).

Java'nın Başarılı Olmasındaki Sebepler

■ Nitelikli bir programlama dili olması

- C++ da olduğu gibi bellek problemlerinin olmaması .
- Nesneye yönelik (Object - Oriented) olması
- C/C++/VB dillerinin aksine dinamik olması .
- Güvenli olması .
- Internet uygulamaları için elverişli (**Applet, JSP, Servlet, EJB, Corba, RMI**).

■ Platform bağımsız olması : bir kere yaz her yerde çalıştır

Çöp Toplayıcı (*Garbage Collector*)

- Bir programın çalışma durumunda ortaya çıkan ve sonradan kullanılmayan (gereksiz) nesneleri bulur ve onları yok eder (*destroy*).
- Bellek yönetiminin (*memory management*) yükü, kodu yazan kişiden Java'ya geçmiş olur
- Diğer dillerde, örneğin C++ da , oluşturulan nesnelerin yok edilme sorumluluğu kodu yazan kişiye aittir.
- Çöp toplayıcısı(*garbage collector*) JVM'in yazılışına (*implementation*) göre değişkenlikler gösterebilir.

Java'da Yorum Satırı

- Java kaynak kodunun içerişine istediğiniz yorumları yazabilmeniz için belli yol izleminiz gerekmektedir.
- Java'da yorum satırlarını belirtme iki şekilde mümkün olur
 1. **/* yorum */** , slash - yıldızdan , diğer yıldız-slash arasında kadar istediğiniz yorumu yazabilirsiniz . Uzun satırlı yorumlarda bu yöntemi kullanabilirsiniz.
 2. **// yorum** , tek satırlık yorum yapmak için idealdir. Kısa yorularınız için bu yöntemi kullanabilirsiniz.

Hersey Nesne - 1

- Java'da herşeye nesne olarak davranırız. Herseyin nesne olmasına rağmen nesneleri yönetmek için “referanslar” kullanılır .

Örnek : Diyelim ki elimizde bir maket uçak (nesne olarak düşünün) ve bu maket uçağa ait bir de kumanda (referans) olduğunu düşünelim.

Bu maket uçağı havada sağa sola döndürmek için elimizdeki kumanda cihazını kullanmak zorundayızdır; benzer şekilde havalandırmak veya yere indirmek için de kumanda cihazından faydalananız. Burada dikkat edilmesi gereken unsur kumanda cihazından çıkan emirlerin maket uçağı tarafından yerine getirilmesidir.

Herşey Nesne - 2

- Elimizde uzaktan kumandanın (referans) olması, maket uçağımızın (nesne) olduğu anlamına gelmez .
- Uzaktan kumandamız (referans) da tek başına hayatı sürdürbilir.

```
String kumanda ; // kumanda referansı şu an için  
//String bir nesneye bağlı değil.
```

String Kumanda ;

String tipinde bir referans,
su an için herhangi bir
String nesnesine bağlı
değil

Herşey Nesne - 3

- Bir referansa mesaj göndemek istiyorsak onu bir nesneye bağlamamız gereklidir.

```
•String kumanda= new String("Selamlar") ;
```

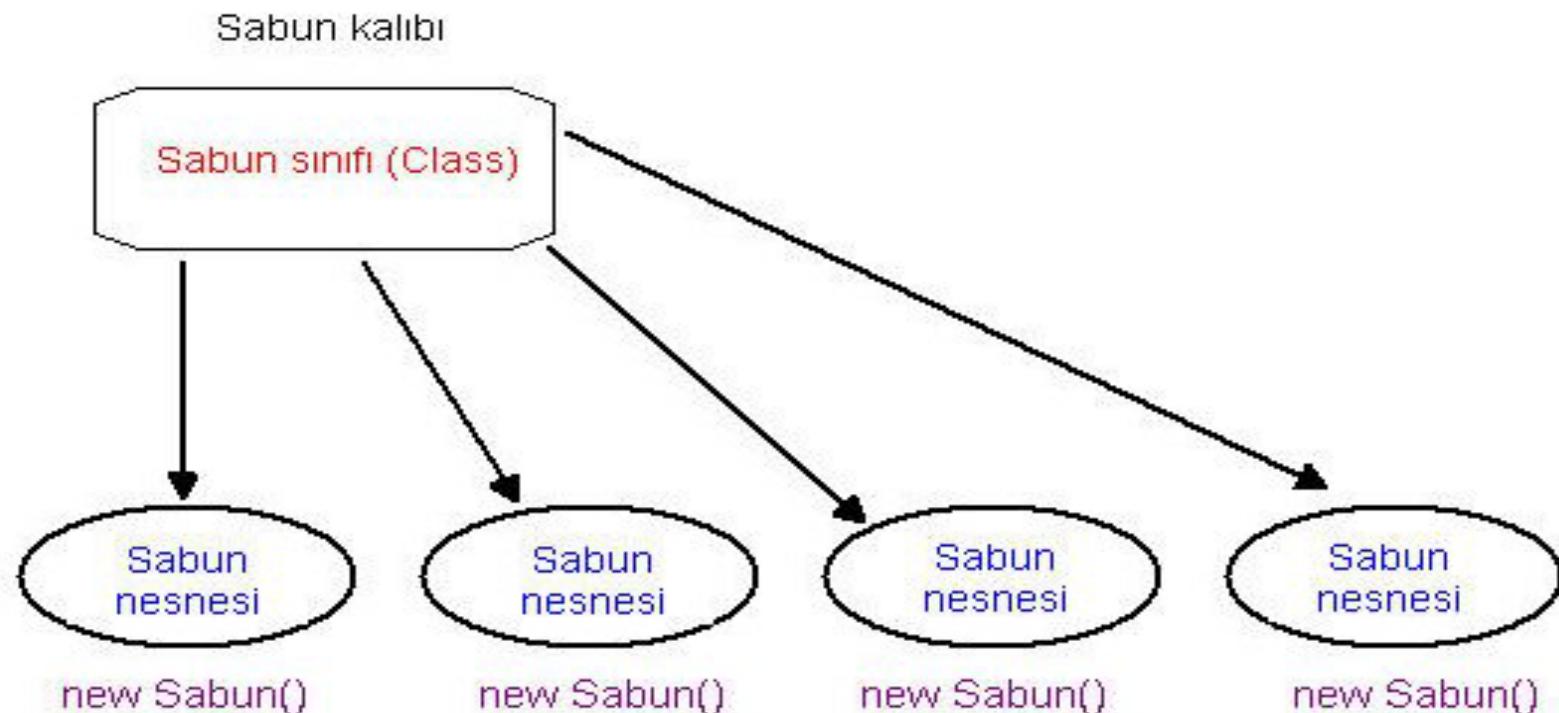
```
•String kumanda="Selamlar" ;
```



String tipinde bir referans,
artık bir String nesnesine
bağlı !!

String nesnesi

Sınıf Nedir? Nesne Nedir?



Sabun s = new Sabun();

Java'da Depolanan Veriler Nerede Durur - 1

- **Stack** : Bulunduğu nokta RAM'dır... Stack üzerinde referansların kendileri bulunur.
- **Heap** : Burası genel amaçlı bir havuzdur . Nesnelerin kendisi bu alanda durur.
- **Statik Alan** : Bu alan RAM'de bulunur. Statik alanda yer alan veriler , programın çalışması süresince orada yaşarlar. Nesnelerin kendileri bu alanda yer almazlar.

Java'da Depolanan Veriler Nerede Durur - 2

- **Non-RAM Bellek** : Bazı durumlarda uygulamaların içerisinde oluşturduğumuz nesnelerin, uygulama sonlandıktan sonra bile varlıklarını sürdürmelerini isteriz.
 1. **Akışkan Nesneler (streamed objects)** : Bu nesneler, genellikle ağ(network) üzerindeki başka bir makineye gönderilmek üzere bayt ırmaklarına dönüştürülürler.
 2. **Kalıcı Nesneler (persistent objects)** : Bu nesneler kendi durumlarını(*state*) saklarlar ve diskimizde saklanırlar. Kendi durumlarını saklamaktan kasıt ise özelliklerinin (*attribute*) değerlerinin korunmasıdır.

Temel (*Primitive*) Tipler

- Temel tipler stack alanında saklanırlar.

Temel tip	Boyut	Minimum	Maximum	Sarmalıyıcı Sınıf Tipi
boolean	–	–	–	Boolean
char	16- bit	Unicode 0	Unicode $2^{16}-1$	Character
byte	8- bit	-128	+127	Byte
short	16- bit	-2^{15}	$+2^{15}-1$	Short
int	32- bit	-2^{31}	$+2^{31}-1$	Integer
long	64- bit	-2^{63}	$+2^{63}-1$	Long
float	32- bit	IEEE754	IEEE754	Float
double	64- bit	IEEE754	IEEE754	Double
void	–	–	–	Void

Sarmalayıcı (*Wrapper*) Sınıflar

- Temel tiplerin birer adet sarmalayıcı (*wrapper*) sınıfları bulunur.

```
char c = 'x' ; // temel tip
```

```
Character C = new Character(c) ; // sarmalayıcı sınıf
```

Geçerlilik Alanı (Scope) - 1

```
{  
    int a = 177;  
  
    /* sadece a mevcut */  
  
    {  
        int b = 196;  
  
        /* a ve b mevcut */  
  
    }  
    /* sadece a mevcut */  
    /* b "geçerlilik alanının dışına çıktı" */  
}
```

Geçerlilik Alanı (Scope) - 2

- C ve C++ doğru ama Java'da yanlış olan bir ifade

```
{ // dış alan  
  
int a = 12;  
  
{ // iç alan  
  
    int a = 96; /* java'da yanlış ama C ve C++ doğru */  
  
} // iç alanın sonu  
  
} //dış alanın sonu
```

Nesneler İçin Geçerlilik Alanı (*Scope of Objects*)

```
if (true) {  
  
    String s = new String("Selamlar");  
  
} /* geçerlilik alanının sonu */
```

- Geçerlilik alanının sonunda String nesnesi “Çöp Toplayıcısı” (*Garbage Collector*) tarafından bellekten silineceektir.

Yeni Sınıf Oluşturma

```
public class YeniBirSinif {  
    . . . . .  
}
```

Alanlar - 1

- Alanlar, temel bir tip veya sınıf tipinde olabilir.

```
public class YeniBirSinif {  
    public int i;  
    public double d;  
    public boolean b;  
}
```

Alanlar - 2

Temel (<i>primitive</i>) Tip	Mevcut değer (<i>Default value</i>)
<code>boolean</code>	<code>false</code>
<code>char</code>	<code>'\u0000'</code> (<code>null</code>)
<code>byte</code>	<code>(byte) 0</code>
<code>short</code>	<code>(short) 0</code>
<code>int</code>	<code>0</code>
<code>long</code>	<code>0L</code>
<code>float</code>	<code>0.0f</code>
<code>double</code>	<code>0.0d</code>

Alanlar - 3

```
public class YeniBirSinif  {
    public int i = 5 ;
    public double d = 3.23;
    public boolean b = true ;
}
```

Alanlar - 4

```
YeniBirSinif ybs = new YeniBirSinif();
```

Alanlara Ulaşım

- Nesnenin alanlarına ulaşmak için “.” (nokta) kullanılır.
- Bu alanların erişim belirleyicileri
 - public
 - private
 - protected
 - friendlyolabilir.

```
YeniBirSinif    ybs = new YeniBirSinif();  
ybs.i ;  
ybs.d ;  
ybs.b ;
```

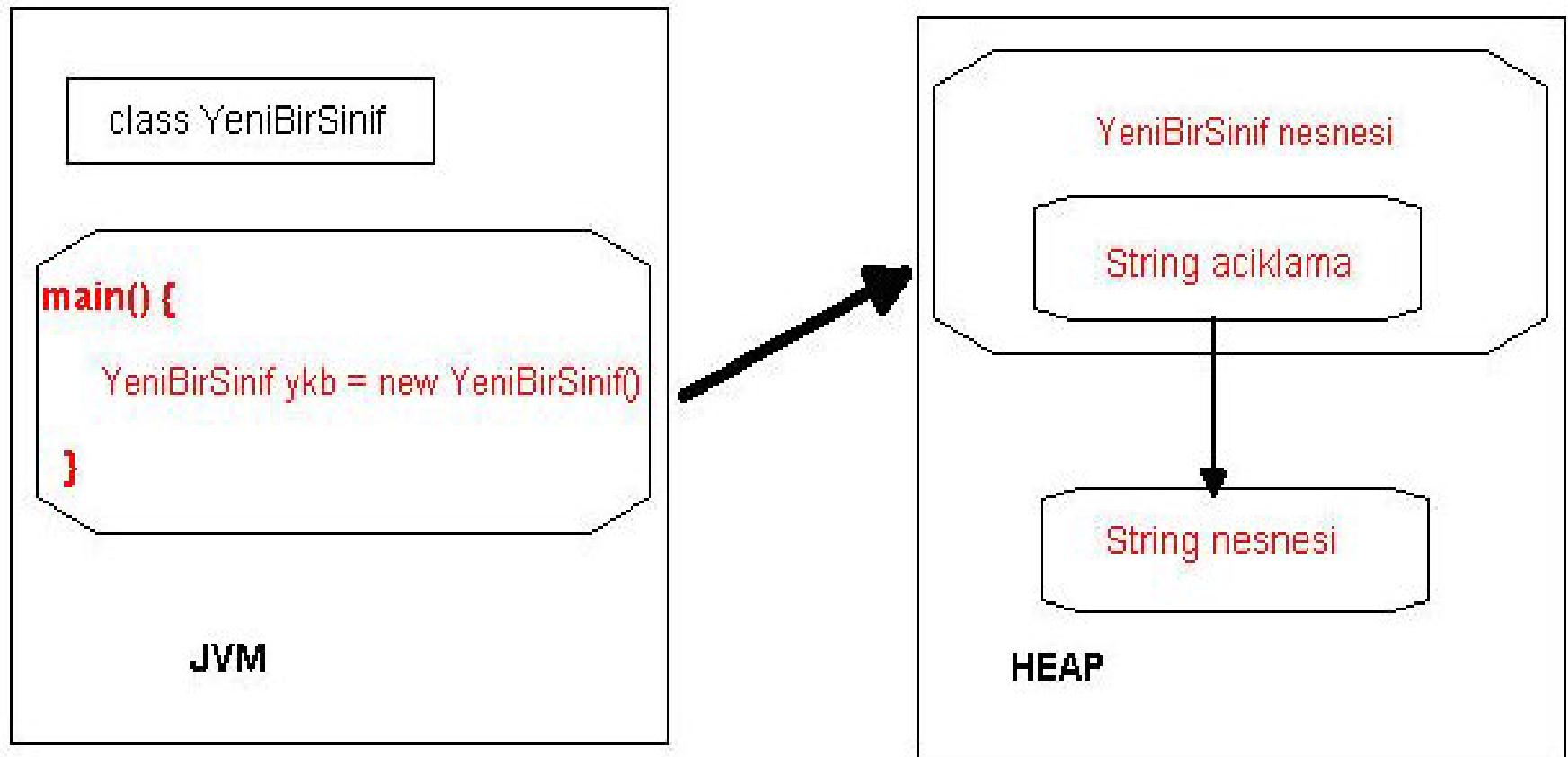
Alanlara Değer Atama

```
YeniBirSinif ybs = new YeniBirSinif();  
ybs.i = 5;  
ybs.d = 5.3;  
ybs.b = false;
```

Sınıf Tipindeki Alanlar - 1

```
public class YeniBirSinif {  
  
    public int i;  
    public double d;  
    public boolean b;  
    public String aciklama = new String("aciklama");  
  
}
```

Sınıf Tipindeki Alanlar - 2



Yordamlar (*Methods*) - 1

```
dönüşTipi Yordamİsmi( /* parametre listesi */ ) {  
    /* Yordamın gövdesi */  
}
```

Yordamlar (*Methods*) - 2

■ **dönüşTipi** = Yordamların iki şansı vardır:

- Değer döndürürler
 - Temel (*primitive*) bir tipde değer (int, double, short vb..)
 - Sınıf tipinde bir değer (String, Double, Short vb...)
- Değer döndürmezler = **void**

Yordamlar (*Methods*) - 3

- **yordamİsmi** = Java'nın kendisine ait olan sözcükler (**if** , **else** , **import** , **class** , **return**..vb) ve Türkçe karakterler haricinde istenilen isim kullanılabilir. Ancak, yordamlar bir eylem içerdikleri için, yordam isimlerinin de bir eylemi belirtmesi tercih edilir.
- Örneğin:
 - **sayiSirala()**
 - **enBuyukSayiBul()**
 - **sqlCalistir()**

Yordamlar (*Methods*) - 4

- **parametre listesi**= Yordam içerisinde işlemler yapabilmek için gerekli olan parametreler. Bu parametreler temel tipte veya sınıf tipinde olabilirler.

Yordamlar (*Methods*) - 5

- **Yordam gövdesi** = Bu kısım kodu yazan kişinin yaratıcılığına bağlı olarak değişir.

Yordam (*Method*) Örneği - 1

```
int boyutDondur(String kelime) {  
    return kelime.length();  
} // yordamın sonu
```

Yordam (*Method*) Örneği - 2

```
String elmaHesapla(int elmasayisi) {  
    return new String("elma sayisi = "  
                      + elmasayisi*2);  
} // yordamın sonu
```

Yordam İçindeki Yerel Değişkenlerin İlk Değerlerini Alması

```
void hesapla(String kelime , int kdv ) {  
    int sondeger = 0;  
    int kelimeboyut = 0 ;  
    int toplamboyut ; // Hatalı !!  
    toplamboyut++ ; // Hatalı !!  
    kelimeboyut = kelime.length();  
    sondeger = kelimeboyut + kdv ;  
}
```

İlk Java Programı - 1

```
public class Selam {  
    public static void main(String args[]) {  
        System.out.println("Selamlar !");  
    }  
}
```

İlk Java Programı - 2

- **public class Selam** : Bu kısım da yeni bir sınıf oluşturuyor...

İlk Java Programı - 3

- **public static void main(String args[])**
 - Java'da bir sınıfın tek başına çalışması isteniyorsa (*standalone*) bu yordam yazılmak zorundadır. Bu yordam sınıflar için bir başlangıç noktasıdır.
 - **static** yordamlar nesneye bağımlı olmayan yordamlardır. Bu yordamı kullanmak için, ilgili sınıf'a ait bir nesne oluşturma zorunluluğu yoktur.

Diziler (Arrays)

- **main()** yordamı parametre olarak *String* sınıfı tipinde dizi alır, bu *String* sınıfı tipindeki dizinin içerisinde, konsoldan Java uygulamasına gönderilen parametreler bulunur .
 - args [0] : konsoldan girilen 1. parametre değerini taşır ...
 - args [1] : konsoldan girilen 2. parametre değerini taşır ...
 - args [n-1] : konsoldan girilen n. parametre değerini taşır ...
- Java'da diziler sıfır'dan başlarlar. Diziler ilerleyen bölümlerde yoğun bir şekilde incelenecektir.

- **System.out.println("Selamlar !")**
 - Bu komut satırı, bilgileri konsola (ekrana) basmamızı sağlar. Java'nın dokümanlarına bakarsak;
 - **System** sınıfı altında static bir alan olan **out** alanının mevcut olduğunu görüyoruz. Bu yüzden **System** sınıfını oluşturmak zorunda değiliz (**new System()**).
 - **out** alanı bize **PrintStream** nesnesi oluşturur ve **PrintStream** nesnesinin **println()** methodu ile bilgileri konsola(ekrana) bastırırız.

Kurulum

- Kurulumlar dökümanlardan incelenebilir.

Dizin Yapısı

JAVA_KURULUM_DIZINI

```
|__bin (dizin)  
|__demo (dizin)  
|__include (dizin)  
|__jre  
|   |__bin(dizin)  
|   |__lib (dizin)  
|       |__rt.jar (class dosyalarının bulunduğu jar dosyası)  
|__lib (dizin)  
|   |__tools.jar (Faydalı sınıfların bulunduğu jar dosyası)  
|__src.jar (Kaynak kodların bulunduğu jar dosyası)
```

Nedir bu `args[]`? Ne İşe Yarar?

```
public class ParametreUygulamasi {  
    public static void main(String[] args) {  
        System.out.println("Girilen Parametre = "+args[0]);  
    }  
}
```

```
bash#     javac ParametreUygulamasi.java
```

```
bash# java ParametreUygulamasi test
```

Girilen Parametre = test

Hata Durumu

```
public class ParametreUygulamasi {  
    public static void main(String[] args) {  
        System.out.println("Girilen Parametre = "+args[0]);  
    }  
}
```

```
bash#     javac ParametreUygulamasi.java
```

```
bash# java ParametreUygulamasi
```

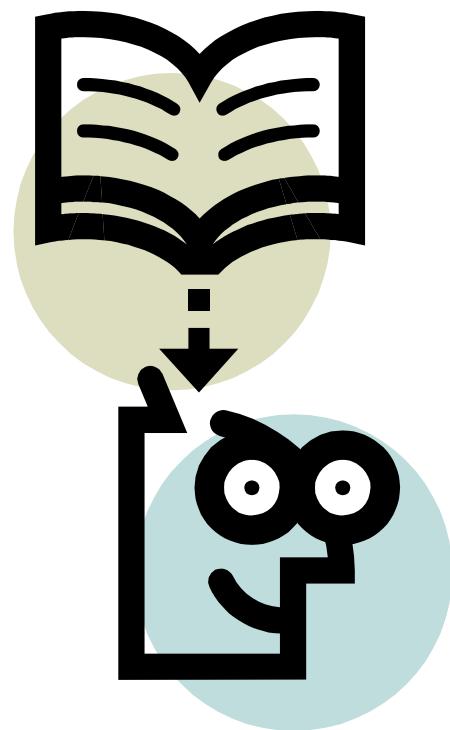
```
Exception in thread "main"  
java.lang.ArrayIndexOutOfBoundsException at  
ParametreUygulamasi2.main(ParametreUygulamasi.java:3)
```

Javadoc – Yorum İle Dökümantasyon Oluşturmak

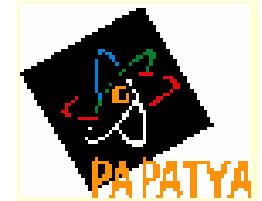
- Dökümantasyon oluşturma yükünü ortadan kaldırır.
- Oluşturulan dökümanlar HTML formatında olur.

```
/** Ilk Java Kodumuzun Dokumantasyonu
Ekrana Selamlar diyen bir uygulama
* @author Altug B. Altintas (altuga@kodcu.com)
* @version 1.0
* @since 09.01.2002
*/
public class SelamDoc {
    /**sayiyi artirmak icin ,
     *degiskenler icin bir ornek
    */
    public int sayac = 0 ;
    /** siniflarda & uygulumalarda giris
     * noktasi olan yordam
     * @param args disaradan girilen
     * parameterler dizisi
     * @return donen deger yok
     * @exception Hic istisna firlatilmiyor
    */
    public static void main(String[] args) {
        System.out.println("Selamlar !");
    }
}
```

Sorular ...



Java'da Program Denetimi ve Operatörler



Atamalar

```
int a ;  
a=4 ; // doğru bir atama  
4=a ; // yanlış bir atama!
```

Temel (Primitive) Tiplerde Atama

```
int a, b ;  
a=4 ;  
b=5 ;  
a=b ;
```

Sonuç : a=5, b=5

Nesneler ve Atamalar

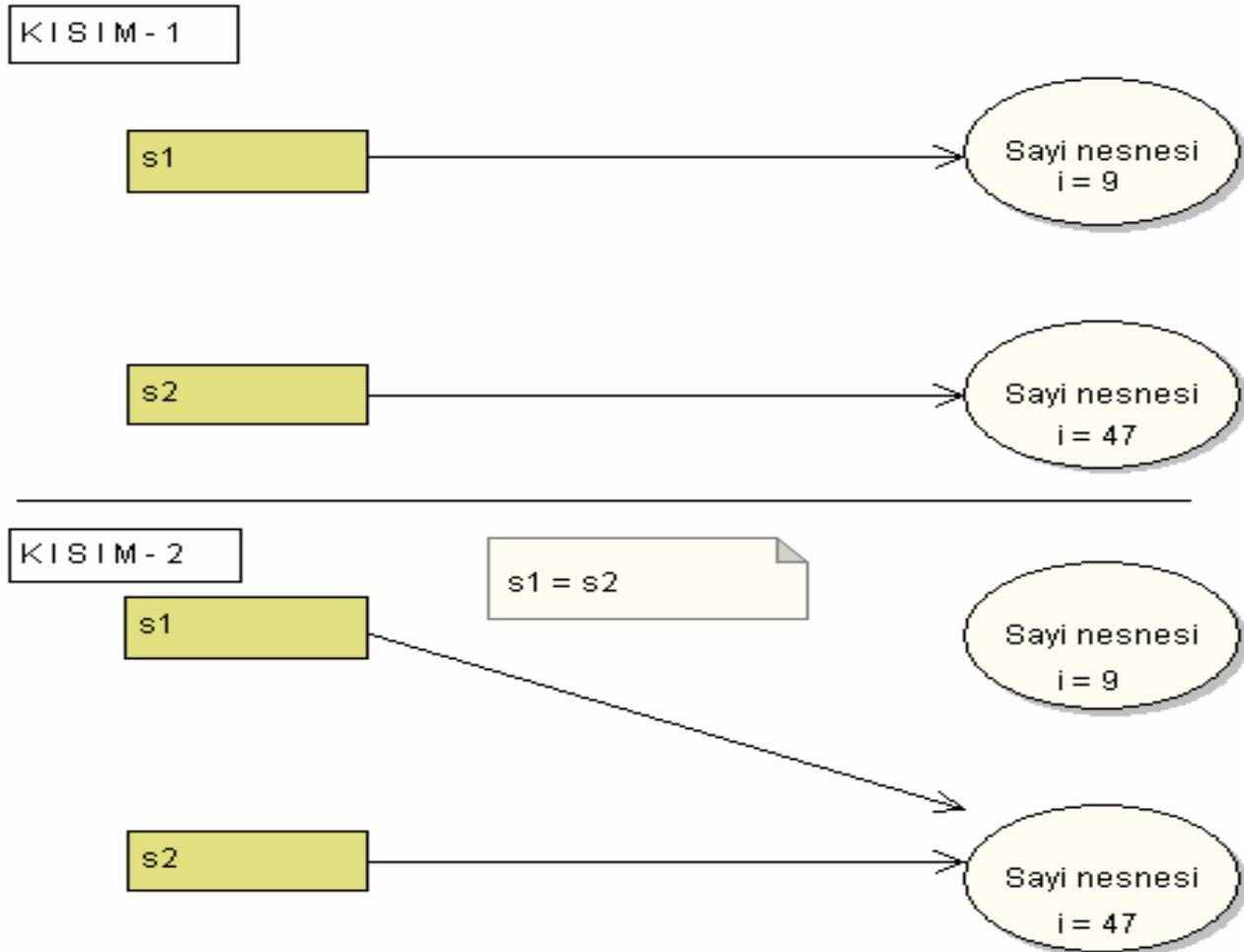


NesnelerdeAtama.java

Sonuç

- 1: **s1.i: 9, s2.i: 47**
- 2: **s1.i: 47, s2.i: 47**
- 3: **s1.i: 27, s2.i: 27**

Şekil



Dosya İsimleri

- Fiziksel dosya ismi ile **public** sınıfın ismi aynı olmalı.

Yordam (Method) Çağırımları

- Yordamlar parametre alırlar.
- Alınan bu paremetreler ile yordam içerisinde işlemler gerçekleşir.
- Peki yordamlara parametre olarak ne gitmektedir ?
 - Nesnenin kendisi mi ?
 - Yoksa nesneye bağlı referans mı ?



IlkelPas.java



Pas.java

java Operatörleri

- Operatörler programlama dillerinin en temel işlem yapma yeteneğine sahip simgesel isimlerdir.
 - Aritmetik Operatör
 - İlişkisel Operatör
 - Mantıksal Operatörler
 - Bit düzeyinde (*bitwise*) Operatörler

java Operatörleri

- Operatörler bir veya daha fazla değişken üzerinden işlemler gerçekleştirirler.
 - İşlem gerçekleştirmek için tek bir değişkene ihtiyaç duyan operatörlere tekli operatör (unary operator)
 - İşlem gerçekleştirmek için iki değişkene ihtiyaç duyan operatörlere ikili operatör (binary operator)
 - İşlem gerçekleştirmek için üç adet değişkene ihtiyaç duyan operatörlere ise üçlü operatör (ternary operator) denir (bir adet var).

Aritmetik Operatörler

Operatör	Kullanılış	Açıklama
+	değişken1 + değişken2	değişken1 ile değişken2 yi toplar
-	değişken1 - değişken2	değişken1 ile değişken2 yi çıkarır
*	değişken1 * değişken2	değişken1 ile değişken2 yi çarpar
/	değişken1 / değişken2	değişken1 ,değişken2 tarafından bölünür
%	değişken1 % değişken2	değişken1 in değişken2 tarafından bölümünden kalan hesaplanır.



AritmetikOrnek.java

“+” ve “-” Operatörleri

Operatör	Kullanılış Şekli	Açıklama
+	+ değişken	Eğer değişken <i>char</i> , <i>sekizli (byte)</i> veya <i>short</i> tipinde ise <i>int</i> tipine dönüştürür.
-	- değişken	Değişkenin değerini negatif yapar (-1 ile çarpar).



OperatorTest.java

Dönüştürme (*Casting*) İşlemi

- Bir temel (*primitive*) tip, diğer bir temel tipe dönüştürülebilir, fakat oluşacak değer kayıplarından kodu yazan kişi sorumludur .



IlkelDonusum.java

String (+) Operatörü

- “+” operatörü **String** tiplerde birleştirme görevi görür.
- Eğer bir ifade **String** ile başlarsa , onu takip eden tiplerde otomatik olarak String nesnesine dönüştürülür.



OtomatikCevirim.java

Uygulamanın Çıktısı

- Sonuc = 012
- **String** bir ifadeden sonra gelen tamsayılar görüldüğü üzere toplantmadı.
- Direk **String** nesnesine çevrilip ekrana çıktı olarak gönderildiler.

Bir Arttırma ve Azaltma

- Java dilinde C dilinde olduğu gibi birçok kısaltmalar vardır.
- Bu kısaltmalar hayatı bazen daha güzel bazen ise çekilmez kılabilir.

Bir Arttırma ve Azaltma Tablosu

Operatör	Kullanılış Şekli	Açıklama
++	değişken++	Önce değişkenin değerini hesaplar sonra değişkenin değerini bir arttırır.
++	++değişken	Önce değişkenin değerini arttırır sonra değişkenin değerini hesaplar.
--	değişken--	Önce değişkenin değerini hesaplar sonra değişkenin değerini bir azaltır.
--	--değişken	Önce değişkenin değerini azaltır sonra değişkenin değerini hesaplar.

Uygulama



OtomatikArtveAz.java

Uygulamanın Çıktısı

```
i      : 1
++i   : 2
i++   : 2
i      : 3
--i   : 2
i--   : 2
i      : 1
```

İlişkisel Operatörler

- İlişkisel operatörler iki değeri karşılaştırarak bu değerler arasındaki mantıksal ilişkiyi hesaplarlar.
- Örneğin iki değer birbirine eşit değilse “**5==8**“
- Bu ilişki çerçevesinde hesaplanan değer **false** olacaktır.

İlişkisel Operatörler Tablosu

Operatör	Kullanılışı	true değeri döner eğer ki...
>	değişken1 > değişken2	değişken1 , değişken2'den büyükse
>=	değişken1 >= değişken2	değişken1 , değişken2'den büyükse veya eşitse
<	değişken1 < değişken2	değişken1 , değişken2'den küçükse
<=	değişken1 <= değişken2	değişken1 , değişken2'den küçükse veya eşitse
==	değişken1 == değişken2	değişken1 , değişken2'ye eşitse
!=	değişken1 != değişken2	değişken1 , değişken2'ye eşit değilse

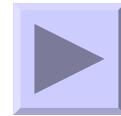
Uygulama



IliskiselDeneme.java

Nesnelerin Karşılaştırılması

- Nesnelerin eşit olup olmadığı ($==$) veya ($!=$) operatörleri ile test edilebilir mi ?



Denklik.java ()*

Uygulamanın Çıktısı

- **false**
- **true**

Uygulama

- Peki bir önceki örneği **Integer** nesneleri yerine temel tip olan **int** tipini kullandık sonuç nasıl olurdu?



IntIcinDenklik.java

Mantıksal Operatörler

- Mantıksal operatörler birden çok karşılaştırma işleminin birleştirip tek bir koşul ifadesi haline getirilmesi için kullanılır.

Mantıksal Operatörler Tablosu

Operatör	Kullanılış Şekli	true değeri döner eğer ki.....
&&	<code>değişken1 && değişken2</code>	Eğer hem değişken1 hemde değişken2 true ise ; (değişken2'yi duruma göre hesaplar*)
	<code>değişken1 değişken2</code>	değişken1'ın veya değişken2'ın true olması ;(değişken2'yi duruma göre hesaplar*)
!	<code>! değişken</code>	Eğer değişken false ise
&	<code>değişken1 & değişken2</code>	Eğer hem değişken1 hemde değişken2 true ise ;
	<code>değişken1 değişken2</code>	değişken1'ın veya değişken2'ın true olması ;
^	<code>değişken1 ^ değişken2</code>	Eğer değişken1 ve değişken2 birbirlerinden farklı ise; ör: değişken1 true , değişken2 false ise*

Uygulama



KosulOp.java

Uygulamanın Çıktısı

```
(a < b)  &&  (c < d)  --> false
(a < b)  ||  (c < d)  --> true
! (a < b)  --> false
(a < b)  &  (c < d)  --> false
(a < b)  |  (c < d)  --> true
(a < b)  ^  (c < d)  --> true
```

Kısa Yollar

- `i = i + 1 ;` yerine.
- `i += 1 ;` kullanılabilir.
- `i = i * 1 ;` yerine
- `i *= 1 ;` kullanılabilir.
-

Kontrol İfadeleri

- Kontrol ifadeleri bir uygulamanın hangi durumlarda ne yapması gerektiğini belirtir.
- Java programlama dilinde toplam 4 adet kontrol ifade çeşidi bulunur.

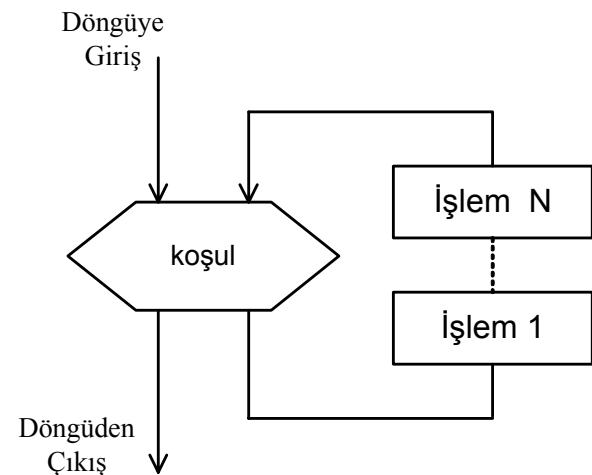
Kontrol İfadeleri Tablosu

İfade Tipi	Anahtar Kelime
Döngü	while , do-while , for
Karar verme	if-else , switch-case
Dallandırma	break , continue , label , return
İstisna yakalama	try-catch-finally , throw

Döngü - while

- **while** ifadesi, çalışması istenen kod bloğunu, durum true ifadesini bulana kadar devamlı olarak çalıştırır.

```
while (koşul) {  
    ...  
    çalışması istenen kod bloğu  
}
```



Uygulama



WhileOrnek.java

Uygulamanın Çıktısı

```
i = 0  
i = 1  
i = 2  
i = 3  
i = 4  
i = 5  
i = 6  
i = 7  
i = 8  
i = 9
```

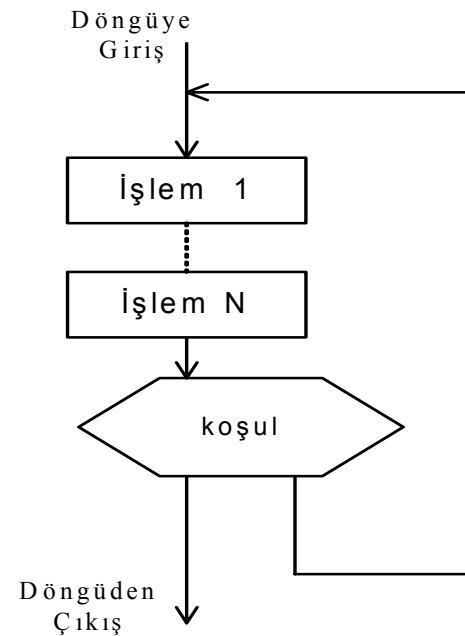
Sayma işlemi tamamlandı.

Döngüleme – **do while**

- **do-while** ifadesi, koşulu en yukarıda değil de en aşağıda hesaplar.
- Böylece **do-while** ifadesinde durum *false* olsa bile çalışması istenen kod bloğuna en az bir kere girilir.



WhileDoOrnek.java



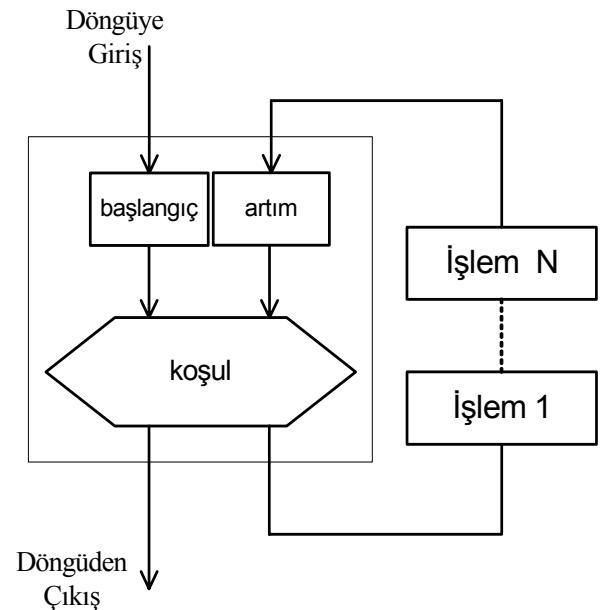
while Döngüsü Kullanırken Dikkat Edilmesi Gereken Hususlar

1. Döngü kontrol değişkenine uygun bir şekilde değer atandığına dikkat edilmeli.
2. Döngü durumunun **true** ile başlamısına dikkat edilmeli.
3. Döngü kontrol değişkeninin uygun bir şekilde güncellendiğinden emin olunması gereklidir (sonsuz döngüye girmemesi için) .

Döngüleme – **for** ifadesi

- Döngünün ne zaman başlayacağı ve ne zaman biteceği en başta belirtilmiştir.

```
for (başlangıç; koşul; artış) {  
    çalışması istenen kod bloğu  
}
```



Uygulama



ForOrnek.java

for İle Sonsuz Döngü

```
for ( ; ; ) {      // sonsuz döngü  
    . . .  
}
```

Uygulamanın Çıktısı

```
i = 0  
i = 1  
i = 2  
i = 3  
i = 4  
i = 5  
i = 6  
i = 7  
i = 8  
i = 9
```

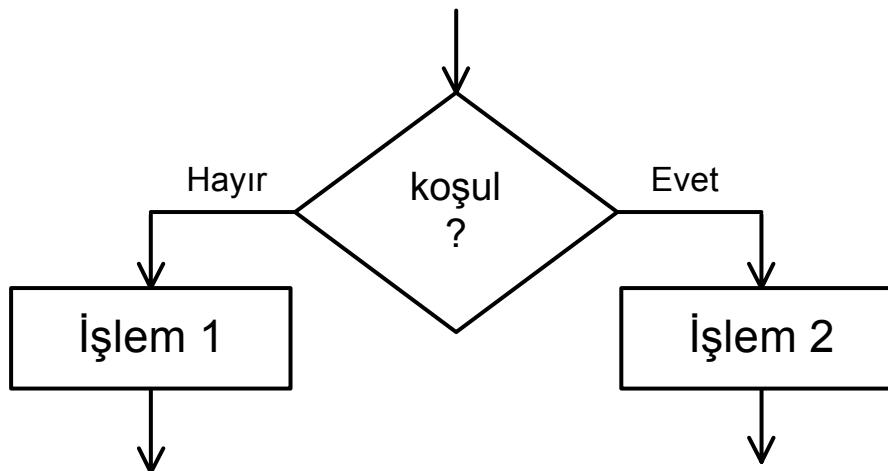
for - Çoklu Değişken

```
public class ForOrnekVersiyon2 {  
  
    public static void main(String args[]) {  
  
        for ( int i = 0 , j = 0 ; i < 20 ; i++ , j++ )  
        {  
            i *= j ;  
            System.out.println("i = " + i + " j = " + j);  
        }  
    }  
}
```

i = 0	j = 0
i = 1	j = 1
i = 4	j = 2
i = 15	j = 3
i = 64	j = 4

Karar Verme - if

```
if (koşul) {  
    durum true olduğunda çalışması istenen kod bloğu  
} else {  
    durum false olduğunda çalışması istenen kod bloğu  
}
```



Uygulama



IfElseTest.java

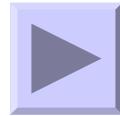
Üçlü if-else

boolean-ifade ? değer0 : değer1

- Eğer *boolean* ifade *true* ise **değer0** hesaplanır , eğer *boolean* ifade *false* ise **değer1** hesaplanır.

Kısa Devre

- **if** ifadesinde eğer **VE(&&)** işlemi kullanılmış ise ve ilk değerden **false** dönmüş ise ikinci değer kesinlikle hesaplanmaz çünkü bu iki değerin sonucunun **VE(And)** işlemine göre **true** dönmesi imkansızdır.
- Kısa devre özelliği sayesinde uygulamalar gereksiz hesaplamalardan kurtulmuş olur.



KısaDevre.java

Karar Verme - switch

```
switch(tamsayı) {  
    case uygun-tamsayı-değer1 : çalışması istenen kod bloğu; break;  
    case uygun-tamsayı-değer2 : çalışması istenen kod bloğu; break;  
    case uygun-tamsayı-değer3 : çalışması istenen kod bloğu; break;  
    case uygun-tamsayı-değer4 : çalışması istenen kod bloğu; break;  
    case uygun-tamsayı-değer5 : çalışması istenen kod bloğu; break;  
    // ...  
    default: çalışması istenen kod bloğu ;  
}
```

Uygulama 1

```
public class AylarSwitchTest {  
  
    public static void main(String[] args) {  
  
        int ay = 8;  
        switch (ay) {  
            case 1: System.out.println("Ocak"); break;  
            case 2: System.out.println("Subat"); break;  
            case 3: System.out.println("Mart"); break;  
            case 4: System.out.println("Nisan"); break;  
            case 5: System.out.println("Mayis"); break;  
            case 6: System.out.println("Haziran"); break;  
            case 7: System.out.println("Temmuz"); break;  
            case 8: System.out.println("Agustos"); break;  
            case 9: System.out.println("Eylul"); break;  
            case 10: System.out.println("Ekim"); break;  
            case 11: System.out.println("Kasim"); break;  
            case 12: System.out.println("Aralik"); break;  
        }  
    }  
}
```

Uygulama 2

```
public class AylarSwitchTestNoBreak {  
  
    public static void main(String[] args) {  
  
        int ay = 8;  
        switch (ay) {  
            case 1: System.out.println("Ocak");  
            case 2: System.out.println("Subat");  
            case 3: System.out.println("Mart");  
            case 4: System.out.println("Nisan");  
            case 5: System.out.println("Mayis");  
            case 6: System.out.println("Haziran");  
            case 7: System.out.println("Temmuz");  
            case 8: System.out.println("Agustos");  
            case 9: System.out.println("Eylul");  
            case 10: System.out.println("Ekim");  
            case 11: System.out.println("Kasim");  
            case 12: System.out.println("Aralik");  
        }  
    }  
}
```

Uygulama 2 - Ekran Çıktısı

Agustos

Eylul

Ekim

Kasim

Aralik

Uygulama 3

```
public class AylarSwitchDefaultTest {  
  
    public static void main(String[] args) {  
  
        int ay = 25;  
        switch (ay) {  
            case 1: System.out.println("Ocak"); break;  
            case 2: System.out.println("Subat"); break;  
            case 3: System.out.println("Mart"); break;  
            case 4: System.out.println("Nisan"); break;  
            case 5: System.out.println("Mayis"); break;  
            case 6: System.out.println("Haziran"); break;  
            case 7: System.out.println("Temmuz"); break;  
            case 8: System.out.println("Agustos"); break;  
            case 9: System.out.println("Eylul"); break;  
            case 10: System.out.println("Ekim"); break;  
            case 11: System.out.println("Kasim"); break;  
            case 12: System.out.println("Aralik"); break;  
            default: System.out.println("Heyooo,Aranilan Kosul" +  
                                "Bulunamadi!!");  
        }  
    }  
}
```

Dallandırma İfadeleri

- Java programlama dilinde dallandırma ifadeleri toplam 3 adettir.
 - **break** ifadesi
 - **continue** ifadesi
 - **return** ifadesi

break İfadesi - Etiketsiz



BreakTest.java

Uygulama Çıktısı

```
i =0  
i =1  
i =2  
i =3  
i =4  
i =5  
i =6  
i =7  
i =8
```

Dongudan çıktı

break İfadesi - Etiketli



BreakTestEtiketli.java

Uygulama Çıktısı

i =0

i =1

i =2

i =3

i =4

i =5

i =6

i =7

i =8

`continue` İfadesi - Etiketsiz



ContinueTest.java

Uygulama Çıktısı

```
•   i =0  
    i =1  
    i =2  
    i =3  
    i =4  
    i =5  
    i =6  
    i =7  
    i =8 → 9 yok  
    i =10  
    i =11  
    i =12  
    i =13  
    i =14  
    i =15  
    i =16  
    i =17  
    i =18  
    i =19  
    i =20  
    i =21  
    i =22  
    i =23  
    i =24  
    i =25  
    i =26  
    i =27  
    i =28  
    i =29
```

Dongudan çıktı

`continue` İfadesi - Etiketli



ContinueTestEtiketli.java

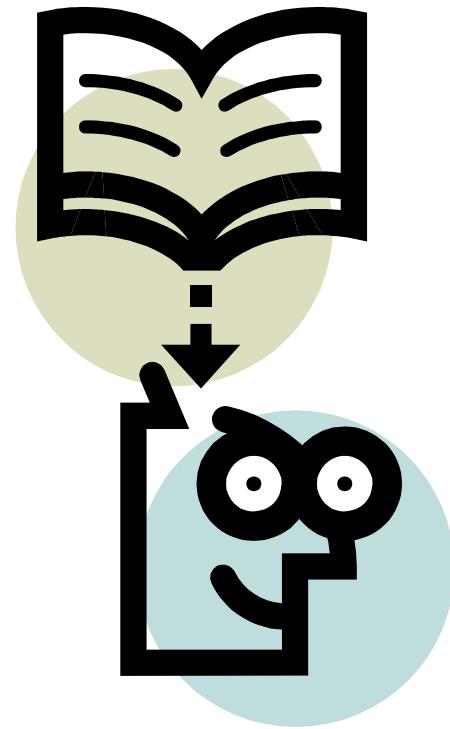
Uygulama Çıktısı

- `i =0`
`i =1`
`i =2`
`i =0`
`i =1`
`i =2`
`i =0`
`i =1`
`i =2`
`i =0`
`i =1`
`i =2`
`i =0`
`i =1`
`i =2`
`i =0`
`i =1`
`i =2`

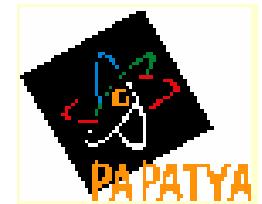
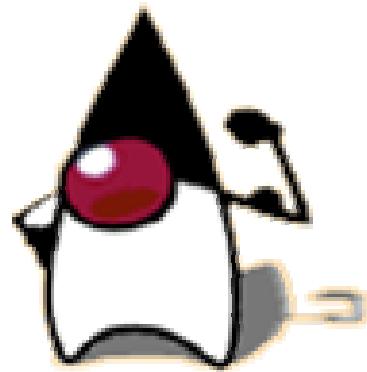
return İfadesi - Etiketli

- Sadece **return** anahtar kelimesi kullanarak yordamların içerisinde tavizsiz bir şekilde terk edebilir.

Sorular ...



Başlangıç Durumuna Getirme ve Temizlik



Hataların sebepleri...

- Nesnelerin yanlış biçimde başlangıç durumlarına getirilmesi
 - Uygulamayı yazan kişi bilmediği kütüphaneye ait nesneleri yanlış şekilde başlangıç durumuna getirmesi nedeniyle hatalarla karşılaşabilir.

Hataların sebepleri

- Temizlik işleminin doğru bir şekilde yapılmaması
 - Oluşturulmuş ve kullanılmayan nesnelerin, sistem kaynaklarında gereksiz yere var olması ile bellek problemleri ortaya çıkabilir.

Başlangıç durumuna getirme işlemi ve yapılandırmacılar

- Bir nesnenin başlangıç durumuna getirilme işlemi (initialization), bir sanatçının sahneye çıkmadan evvelki yaptığı son hazırlık gibi düşünülebilir.
- Oluşturulacak olan nesne kullanıma sunulmadan evvel bazı bilgilere ihtiyaç duyabilir veya bazı işlemleri gerçekleştirmesi gerekebilir (JDBC, konfigurasyon dosyası yüklenmesi gibi).

Yapılandırıcılar (Constructor)

- Yapılandırıcılar içerisinde nesne oluşturulmadan önceki son hazırlıklar yapılır.
- Yapılandırıcılar normal yordamlardan (method) farklıdır.
- Yapıandrıcılar, Java tarafından otomatik olarak çağrırlılar.
- Karşımıza çıkan iki problem
 - *Java Yapılandırıcının ismini nasıl bileyebilir?*
 - *Yapıandrıcının ismi başka yordamların isimleriyle çakışmamalıdır.*

Problemin Çözümü

- Bu problemlere ilk çözüm C++ dilinde bulunmuştur.
- Yapılandırıcının ismi ile sınıf ismi bire bir aynı olmalıdır.
- Böylece Java, yapılandırıcının ismini önceden tahmin edebilecektir.
- İsim karışıklığı minimuma indirgenmiş olur.



YapilandirciBasitOrnek.java

Yapılandırıcılar (Constructor) - 2

- Yapılandırıcılara parametreler aktarılabilir.
- Yapılandırıcı içerisinde herhangi bir şekilde **return** ifadesi ile değer döndürülemez.
(return 5, return true gibi)
- Yapılandırıldan çıkmak istiyorsak sadece return yazılması yeterlidir...



YapilandirciBasitOrnekVersiyon2.java

Adaş Yordamlar (Overloaded Methods)

- İyi bir uygulama yazmak her zaman iyi bir takım çalışması gerektirir.
- Uygulamalardaki yordam (*method*) isimlerinin, yordam içerisinde yapılan iş ile uyum göstermesi önemlidir.
- Bu sayade bir başka kişi sadece yordam ismine bakarak, içerisinde oluşan olayları anlayabilme şansına sahiptir.

Örnek - 1

- Elimizde bulunan

-muzik

-resim

-text

formatındaki dosyaları açmak için yordamlar yazmak istersek, bu yordamların isimlerinin ne olması gereklidir ?

Örnek - 1 (devam)

Yordam isimleri olarak

- muzik dosyası için **muzikDosyasiAc()**
- resim dosyası için **resimDosyasiAc()**
- text dosyası için **textDosyasiAc()**

Örnek - 1 (devam)

- Sonuçta işlem sadece dosya açmaktadır, dosyanın türü sadece bir ayrıntıdır.



MetodOverloadingDemo1.java

Adaş yordamlar nasıl ayırt edilir ?

- Java aynı isimde olan yordamları (overloaded methods) nasıl ayırt edebilmektedir ?
- Her yordamın kendisine özel/tek parametresi veya parametre listesi olmak zorundadır.



MetodOverloadingDemo2.java

Adaş yordamlar dönüş değerlerine göre ayrıt edilebilir mi ?

- Akıllara şöyle bir soru gelebilir : "Adaş yordamlar dönüş tiplerine göre ayrıt edilebilir mi ? "

```
void toplamaYap();
```

```
double toplamaYap();
```

```
double y = toplamayap() ;
```

```
toplamayap() ; // sorun var
```

Varsayılan yapılandırıcılar (*Default constructors*)

- Eğer uygulamamıza herhangi bir yapılandırıcı koymazsak Java bu işlemi kendi otomatik olarak yapmaktadır.
- Varsayılan yapılandırıcılar (parametresiz yapılandırıcılar, default constructor veya "no-args" constructor) içi boş bir yordam olarak düşünülebilir.

Örnek - 2

```
class Kedi {  
    int i;  
}  
  
public class VarsayılanYapilandırıcı {  
    public static void main(String[] args) {  
        //Varsayılan yapılandırıcı çağrıldı  
        Kedi kd = new Kedi();  
    }  
}
```

Örnek - 2 (devam)

```
class Kedi {  
    int i;  
    /* varsayılan yapılandırıcı.  
       Bu yapılandırıcıyı eğer biz koymasaydık  
       Java bizim yerimize zaten koyardı  
    */  
    public Kedi() {}  
  
}  
  
public class VarsayılanYapilandırıcı {  
    public static void main(String[] args) {  
        // varsayılan yapılandırıcı  
        Kedi kd = new Kedi();  
    }  
}
```

Büyünün Bozulması

- Eğer kendimiz yapılandıracı yazarsak, Java bizden varsayılan yapılandırıcı desteğini çekecektir.
- Kendimize ait özel yapılandırmalar tanımlarsak Java'ya *"Ben ne yaptığımı biliyorum, lütfen karışma"* demiş oluruz.



Varsayılan Yapılandırıcı Versiyon2.java

this anahtar kelimesi

- **this** anahtar kelimesi, içinde bulunulan nesneye ait bir referans döner.
- Bu referans sayesinde nesnelere ait global alanlara erişme fırsatı buluruz.



TarihHesaplama.java

Yordam çağrımlarında **this** kullanımı - 2



Yumurta.java

Bir yapılandıricıdan diğerini çağrırmak

- Yapılandırıcı içerisindeki diğer bir yapılandıricıyı çağrıırken **this** ifadesi her zaman ilk satırda yazılmalıdır.
- Her zaman yapılandıricılar içerisindeki **this** ifadesi ile başka bir yapılandıracı çağrılmaz.
- Yapılandırcılar içerisinde birden fazla **this** ifadesi ile başka yapılandıracı çağrılamaz.



Tost.java

Ekran çıktısı

Tost(int sayı ,String malzeme)

Tost(int sayı)

parametresiz yapılandırıcı

Tost sayısı =5 malzeme =Sucuklu

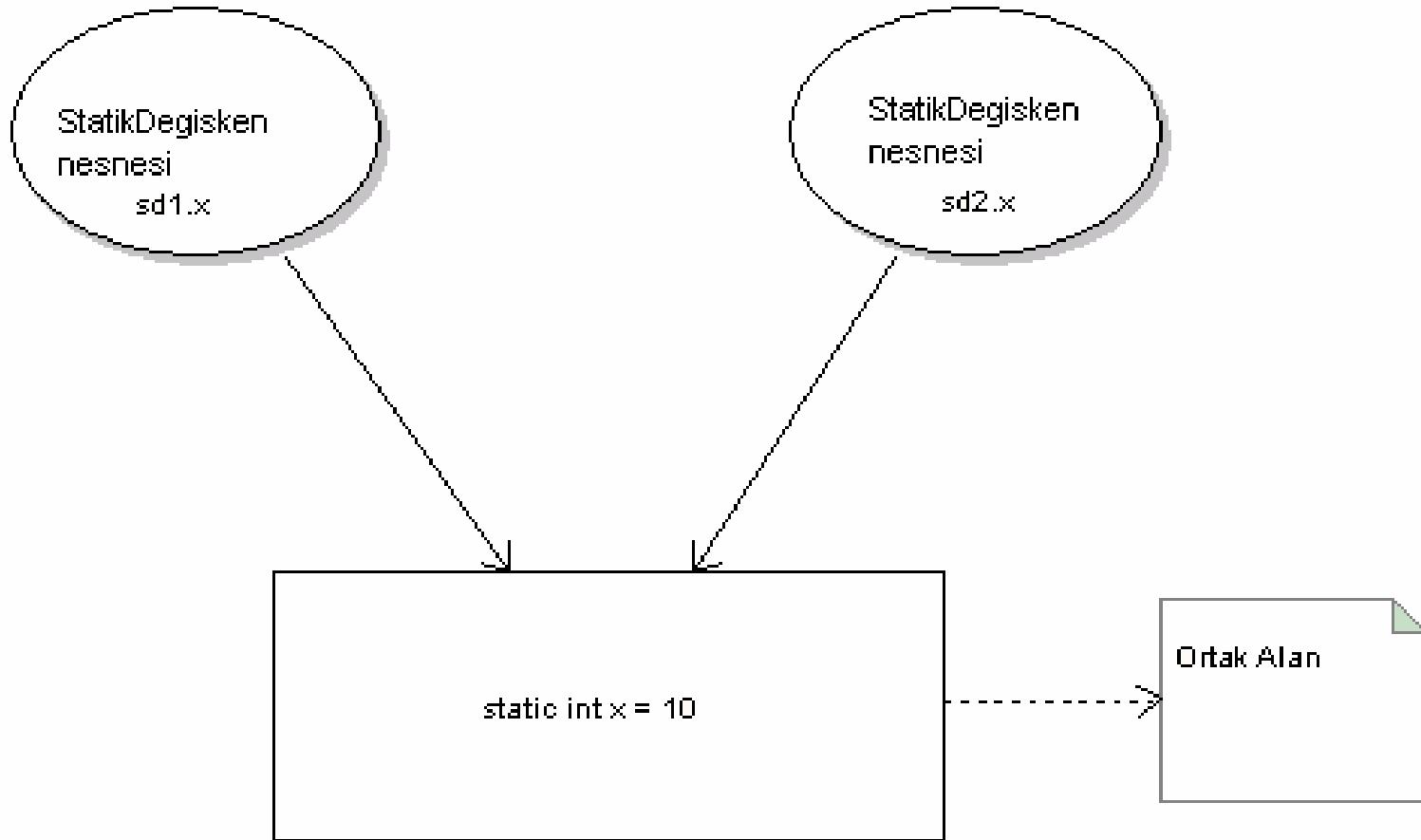
Statik Alanlar (Sınıf Alanları)

- Sadece global olan alanlara statik özelliğini verebiliriz.
- Yerel değişkenlerin statik olma özellikleri yoktur.
- Statik alanlar, bir sınıfı ait olan tüm nesneler için aynı bellek alanında bulunurlar.



StatikDegisken.java

Kuş bakışı görüntü



Statik Yordamlar (*methods*)

- Statik yordamlar (sınıf yordamlar), nesnelerden bağımsız yordamlardır.
- Statik bir yordamı çağrırmak için herhangi bir sınıf'a ait nesne oluşturma zorunluluğu yoktur.
- Statik olmayan yordamlardan (nesneye ait yordamlar), statik yordamları rahatlıkla çağrırlabilmesine karşın statik yordamlardan nesne yordamlarını doğrudan çağrıramayız.



StatikTest.java

Bir yordamın statik mi yoksa nesne yordamı mı olacağına neye göre karar vereceğiz?

- Nesnelerin durumları (state), uygulamanın gidişine göre değişebilir.



MutluAdam.java ()*

Statik yordamlar

- Statik yordamlarlar atomik işler için kullanılırlar.
- Uygulamalarınızda çok fazla statik yordam kullanıyorsanız, tasarımınızı baştan bir kez daha gözden geçirmeniz tavsiye olunur.



Toplama.java

Temizlik İşlemleri: `finalize()` ve çöp toplayıcı (*Garbage Collector*)

- Java dilinde, C++ dilinde olduğu gibi oluşturulan nesnelerimizi işleri bitince yok etme özgürlüğü kodu yazan kişinin elinde değildir.
- Bir nesnenin gerçekten çöp olup olmadığına karar veren mekanizma çöp toplayıcısıdır (*garbage collector*).

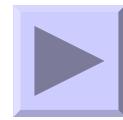
finalize() yordamı

- Akıllarda tutulması gereken diğer bir konu ise eğer uygulamanız çok fazla sayıda çöp nesnesi (kullanılmayan nesne) üretmiyorsa, çöp toplayıcısı (*garbage collector*) devreye girmeyebilir.
- Bir başka önemli nokta;
 - **System.gc()** ile çöp toplayıcısını tetiklemezsek , çöp toplayıcısının ne zaman devreye girip çöp haline dönüşmüş olan nesneleri bellekten temizleneceği bilinemez.

`System.gc()`



Temizle.java



Temizle2.java

```
Elma Objesi Olusturuluyor = 0  
Elma Objesi Olusturuluyor = 1  
Elma Objesi Olusturuluyor = 2  
Elma Objesi Olusturuluyor = 3  
Elma Objesi Olusturuluyor = 4  
Elma Objesi Olusturuluyor = 5  
Elma Objesi Olusturuluyor = 6  
Elma Objesi Olusturuluyor = 7  
Elma Objesi Olusturuluyor = 8  
Elma Objesi Olusturuluyor = 9  
  
Elma Objesi Yok Ediliyor = 0  
Elma Objesi Yok Ediliyor = 1  
Elma Objesi Yok Ediliyor = 2  
Elma Objesi Yok Ediliyor = 3  
Elma Objesi Yok Ediliyor = 4  
Elma Objesi Yok Ediliyor = 5  
Elma Objesi Yok Ediliyor = 6  
Elma Objesi Yok Ediliyor = 7  
Elma Objesi Yok Ediliyor = 8  
Elma Objesi Yok Ediliyor = 9  
  
Elma Objesi Olusturuluyor = 10  
Elma Objesi Olusturuluyor = 11  
Elma Objesi Olusturuluyor = 12  
Elma Objesi Olusturuluyor = 13  
Elma Objesi Olusturuluyor = 14  
Elma Objesi Olusturuluyor = 15  
Elma Objesi Olusturuluyor = 16  
Elma Objesi Olusturuluyor = 17  
Elma Objesi Olusturuluyor = 18  
Elma Objesi Olusturuluyor = 19  
Elma Objesi Olusturuluyor = 20
```

System.gc()
çağrıldaktan sonra çöp
topluyıcısı tetiklendi

Çöp toplayıcısı ,
gerekisiz Elma
objelerini hafızadan
siliyor

Çöp toplayıcısı (Garbage Collector) nasıl çalışır?

- Çöp toplayıcısının temel görevi, kullanılmayan nesneleri bularak bunları bellekten silmektir.
- Sun Microsystems tarafından tanıtılan Java HotSpot VM (Virtual Machine) sayesinde heap bölgesindeki nesneler nesillerine göre ayrılmaktadır.
 - **Eski Nesil**
 - **Yeni Nesil**

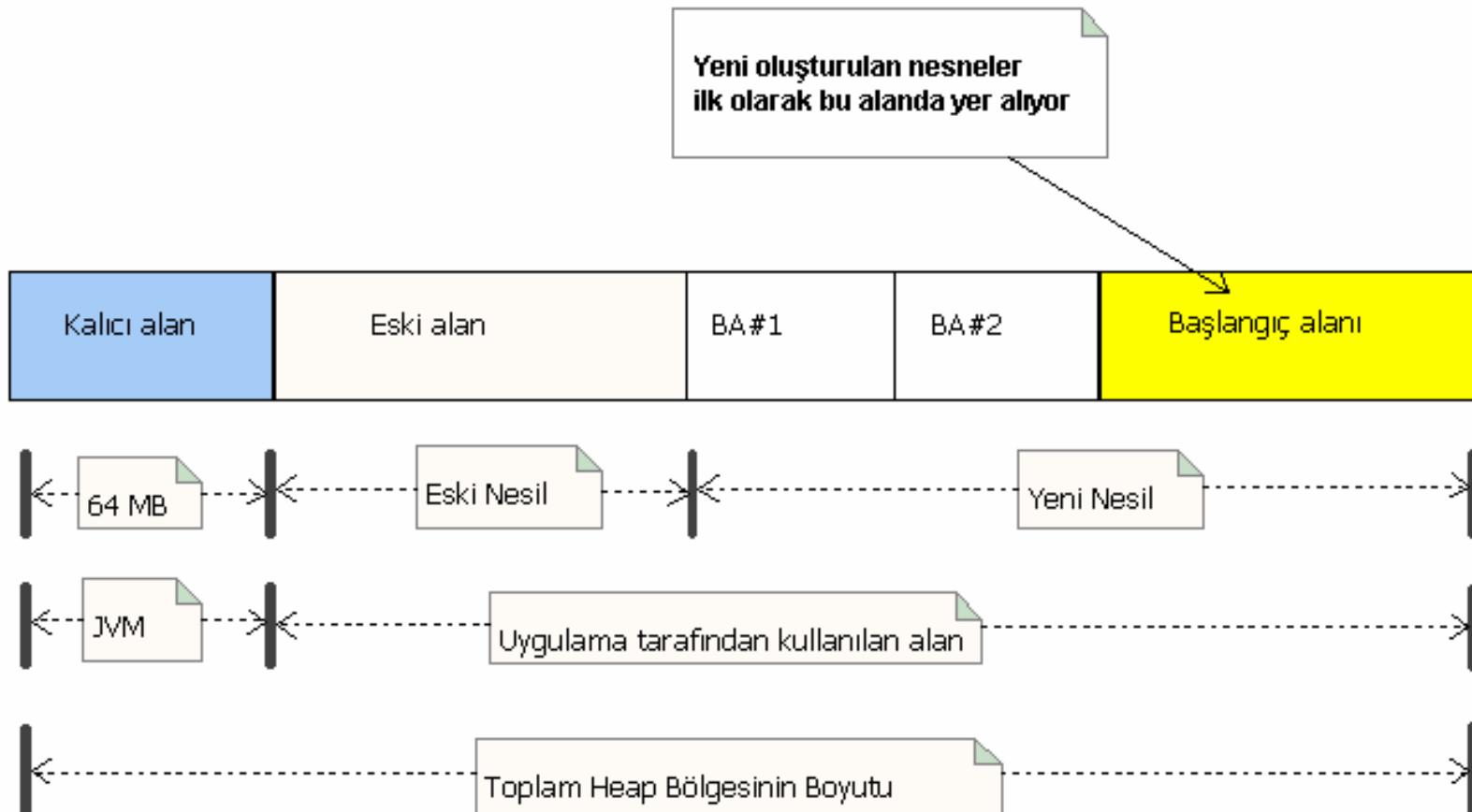
Çöp toplayıcısı (Garbage Collector) nasıl çalışır?

- Nesnelerin bellekten silinmesi görevi kodu yazan kişiye ait değildir.
- Bu görev çöp toplayıcısına aittir. Java 1.3.1 ve daha sonraki Java versiyonları iki noktayı garanti eder;
 - Kullanılmayan nesnelerin kesinlikle bellekten silinmesi.
 - Nesne bellek alanının parçalanmasını engellemek ve belleğin sıkıştırılması.

Çöp toplama teknikleri

- **Eski yöntem**
 - Referans Sayma Yöntemi
- **Yeni Yöntemler**
 - Kopyalama yöntemi (Copy)
 - İşaretle ve süpür yöntemi (Mark and Sweep)
 - Artan (sıra) yöntem (Incremental)

Heap Bölgesine Bakış



Heap bölgesinin boyutları nasıl kontrol edilir.

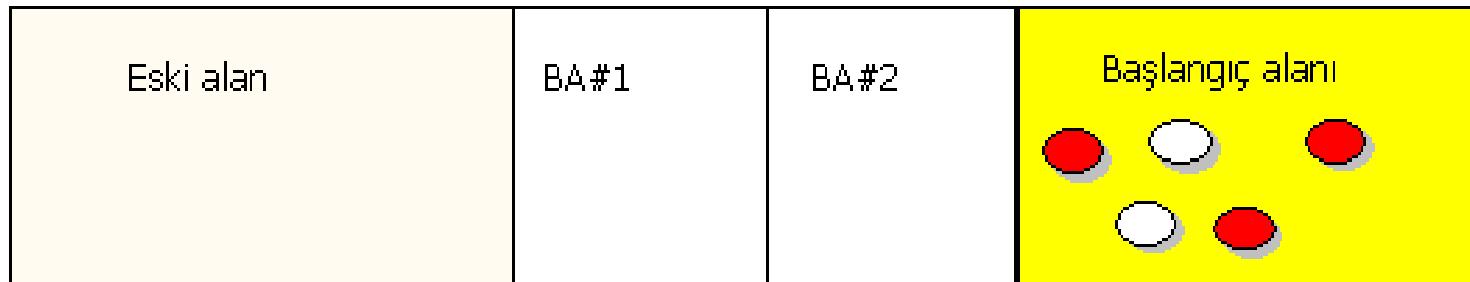
- Heap bölgesine minimum veya maksimum değerleri vermek için **-Xms** veya **-Xmx** parametlerini kullanırız.

```
java -Xms32mb Temizle
```

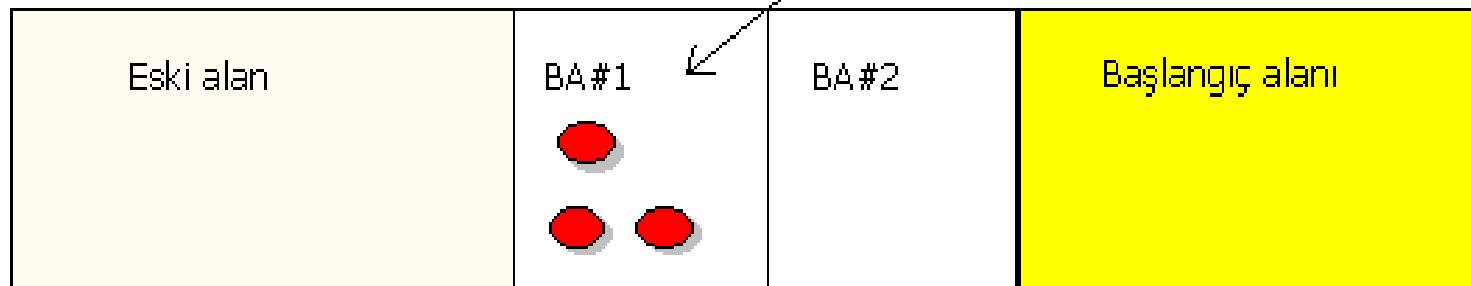
Kopyalama yönteminin gösterimi

- Birazdan gösterilecek olan şeklimizde, **canlı** nesneler kırmızı renk ile ifade edilmiştir.

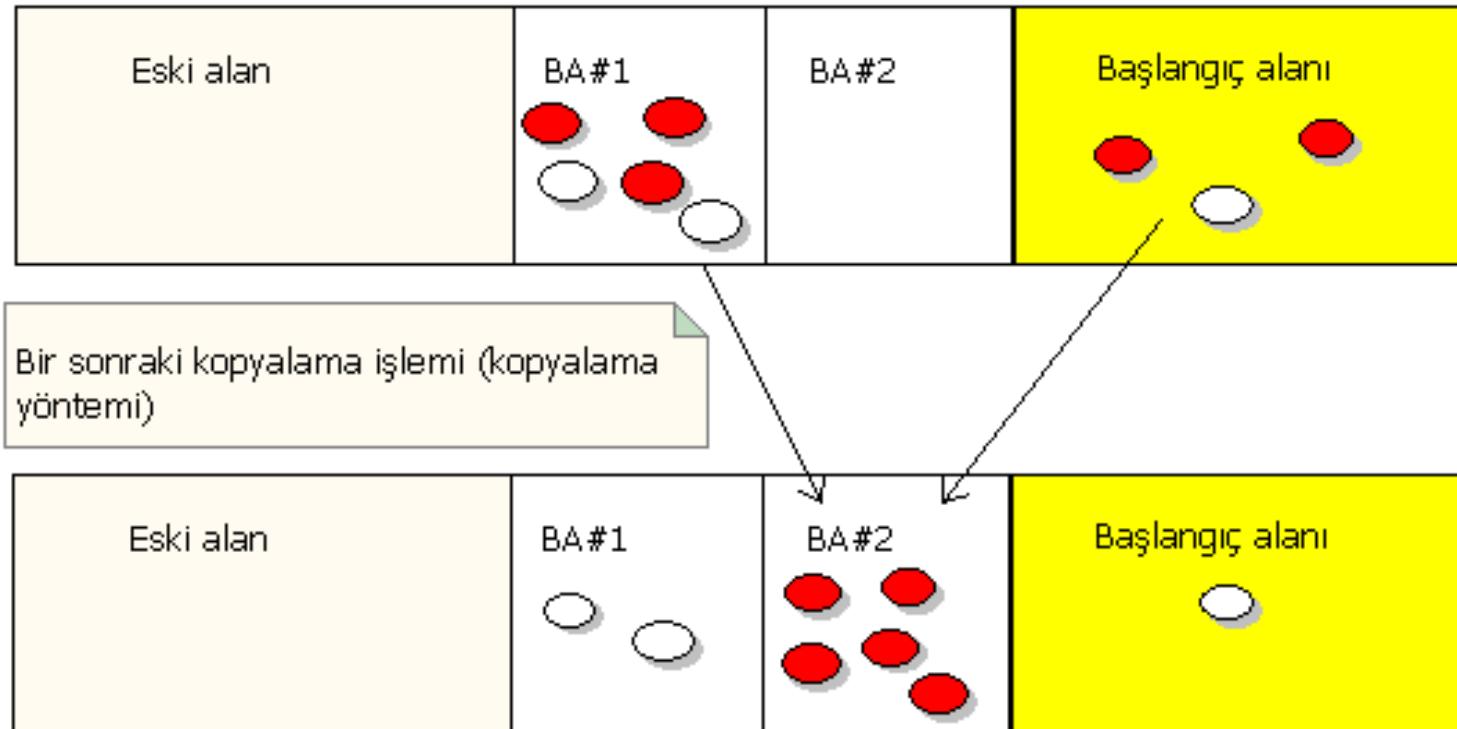
Aşama – 1



Yaşayan objeler boş alana kopyalanır (kopyalama yöntemi)



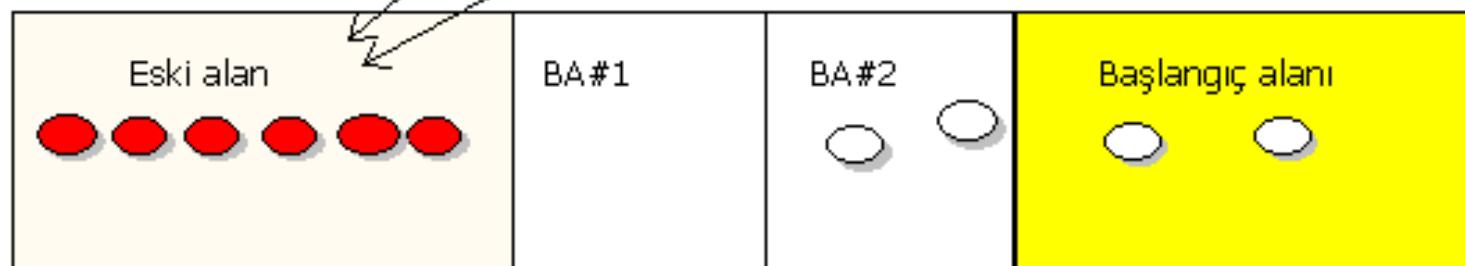
Aşama – 2



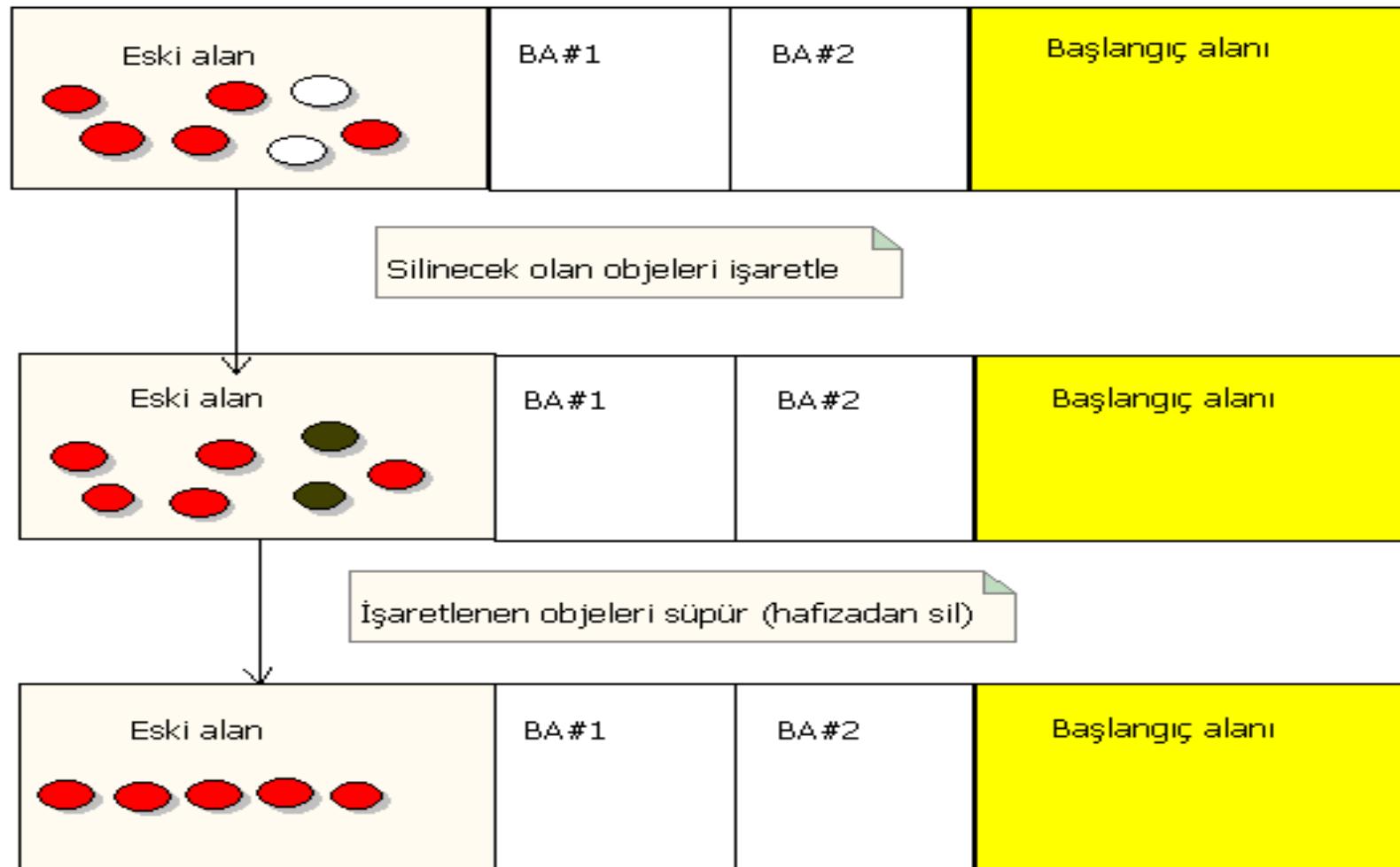
Aşama – 3



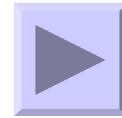
Objeler belli bir olgunluğa ulaşınca eski alana gönderiliyorlar



İşaretle ve süpür yönteminin gösterimi



Kopyala yöntemin ve işaretle ve süpür yöntemi



HeapGosterim.java

```
java -verbosegc HeapGosterim
```

Uygulamanın Çıktısı

```
[Full GC 224K->117K(1984K), 0.0225984 secs]
[GC 629K->117K(1984K), 0.0029275 secs]
[GC 629K->117K(1984K), 0.0010258 secs]
[GC 629K->117K(1984K), 0.0010275 secs]
[GC 629K->117K(1984K), 0.0010334 secs]
[Full GC 179K->117K(1984K), 0.0201336 secs]
[GC 629K->117K(1984K), 0.0009529 secs]
[GC 629K->117K(1984K), 0.0009244 secs]
[GC 629K->117K(1984K), 0.0009412 secs]
[GC 629K->117K(1984K), 0.0009454 secs]
[Full GC 179K->117K(1984K), 0.0202417 secs]

.....
.....
```

İsaretle ve
Süpür yöntemi
calisti

Kopyalama
yöntemi

Alanlara ilk değerleri atama

- Java uygulamalarında üç tür değişken çeşiti bulunur:
 - Yerel (*local*) değişkenler.
 - Nesneye ait global alanlar.
 - Sınıfa ait global alanlar (statik alanlar).

Örnek - 3



DegisenGosterim.java

Yerel Değişkenler

```
public int hesapla () { // yerel değişkenlere ilk değerleri her zaman
                      // verilmelidir.

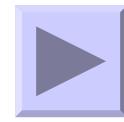
    int i ;

    i++; // ! Hata ! ilk deger verilmeden üzerinde işlem yapılamaz

    return i ;

}
```

Nesneye ait global alanlar – ilkel tipler



IlkelTipler.java

Nesneye ait global alanlar - sınıf tipleri



NesneTipleri.java

Sınıflara ait global değişkenler - ilkel tipler

- **Önemli Nokta:** Statik olan alanlara sadece bir kere değer atanır.



IlkelTipлерStatik.java

Sınıflara ait global değişkenler – sınıf tipleri



StatikNesneTipleri.java

İlk değerleri atarken yordam kullanımı



KarisikTipler.java

İlk değer verme sıralaması

- Nesneye ait global alanlara ilk değer hemen verilir, hatta yapılandırıcıdan bile önce...
- Alanların konumu hangi sırada ise ilk değer verme sıralaması da aynı sırada olur.



Defter.java

Statik ve statik olmayan alanların değer alma sıralaması

- Statik alanlar sınıflara ait olan alanlardır ve statik olmayan (nesneye ait alanlar) alanlara göre ilk değerlerini daha önce alırlar.



Kahvalti.java

Statik alanlara toplu değer atama

- Statik alanlarımıza toplu olarak değer atama.



StatikTopluDegerAtama.java

Statik olmayan alanlara toplu değer atama



NonStatikTopluDegerAtama.java

Diziler (Arrays)

- Diziler nesnedir.
- Dizi nesnesi, içinde belli sayıda eleman bulundurur.
- Dizi içerisindeki ilk elemanın konumu **0**'dan başlar, son elemanın yeri ise **n-1**'dir.

Dizi tipindeki değişkenler

```
double[] dd ; // double tipindeki dizi  
double dd[] ; // double tipindeki dizi  
float [] fd ; // float tipindeki dizi  
Object[] ao ; // Object tipindeki dizi
```

Dizileri oluşturmak

```
double[] d    = new double[20] ;  
double dd[]   = new double[20];  
float [] fd   = new float [14];  
  
Object[] ao   = new Object[17];  
String[] s    = new String[25] ;
```

Dizilerin tekrardan boyutlandırılması

```
int liste[] = new int[5] ;  
  
// yeni bir dizi nesnesine bağlandı  
liste = new int[15] ;
```

Dizi içerisindeki elemalara ulaşım



DiziElemanlariGosterimBir.java

Diziler içerisinde elemanların sıralanması



DiziSiralama.java

Dizilerin dizilere kopyalanması



DizilerinKopyalanmasi.java

Çok Boyutlu Diziler

Dizi içerisinde dizi tanımlanabilir.

```
int[][] t1 = {  
    { 1, 2, 3, },  
    { 4, 5, 6, },  
};
```

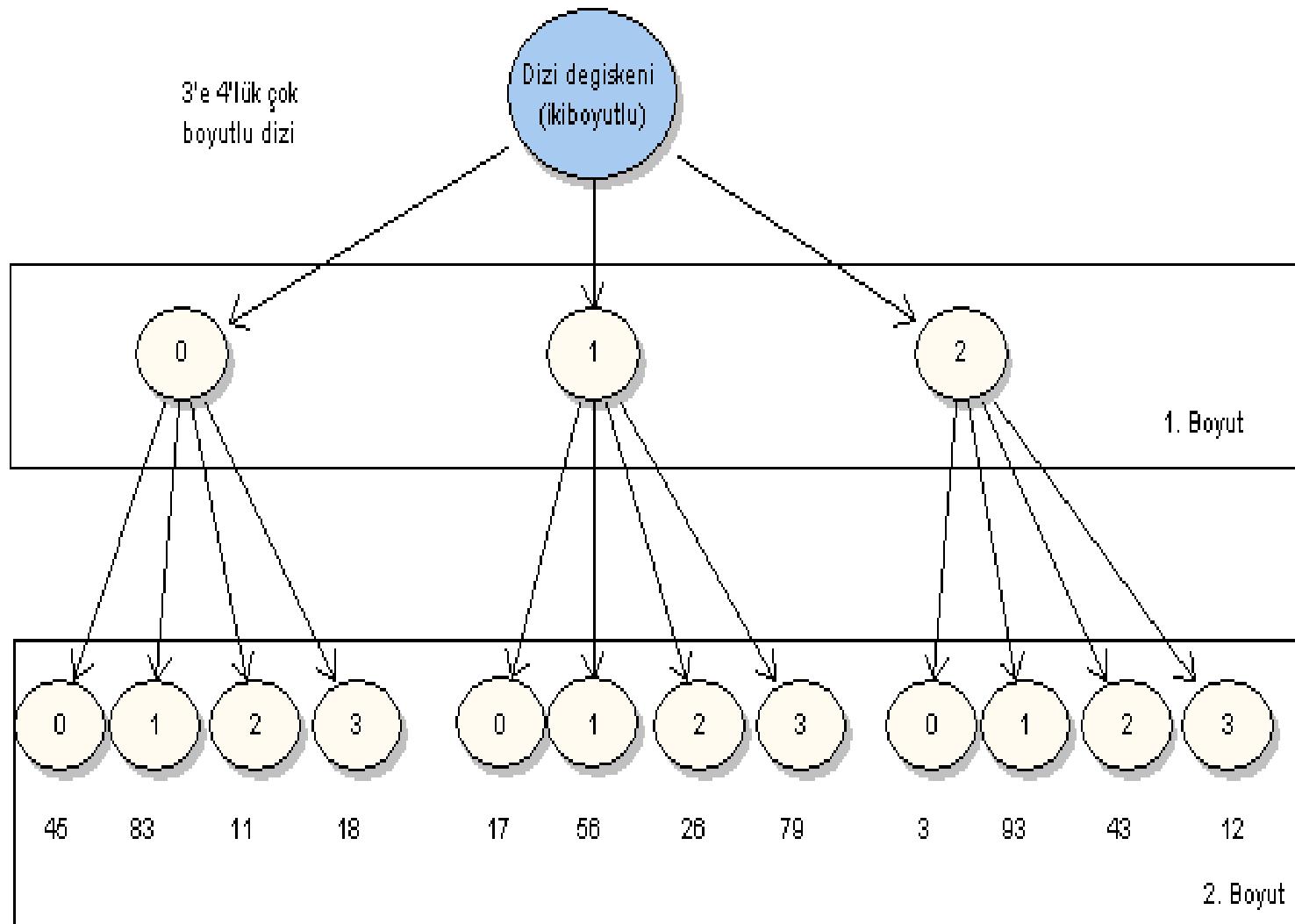
Çok boyutlu dizileri oluşturmanın diğer bir yolu

```
int [][] t1 = new int [3][4] ;
```

```
int [][] t1 = new int [][4] ; //!Hata!
```



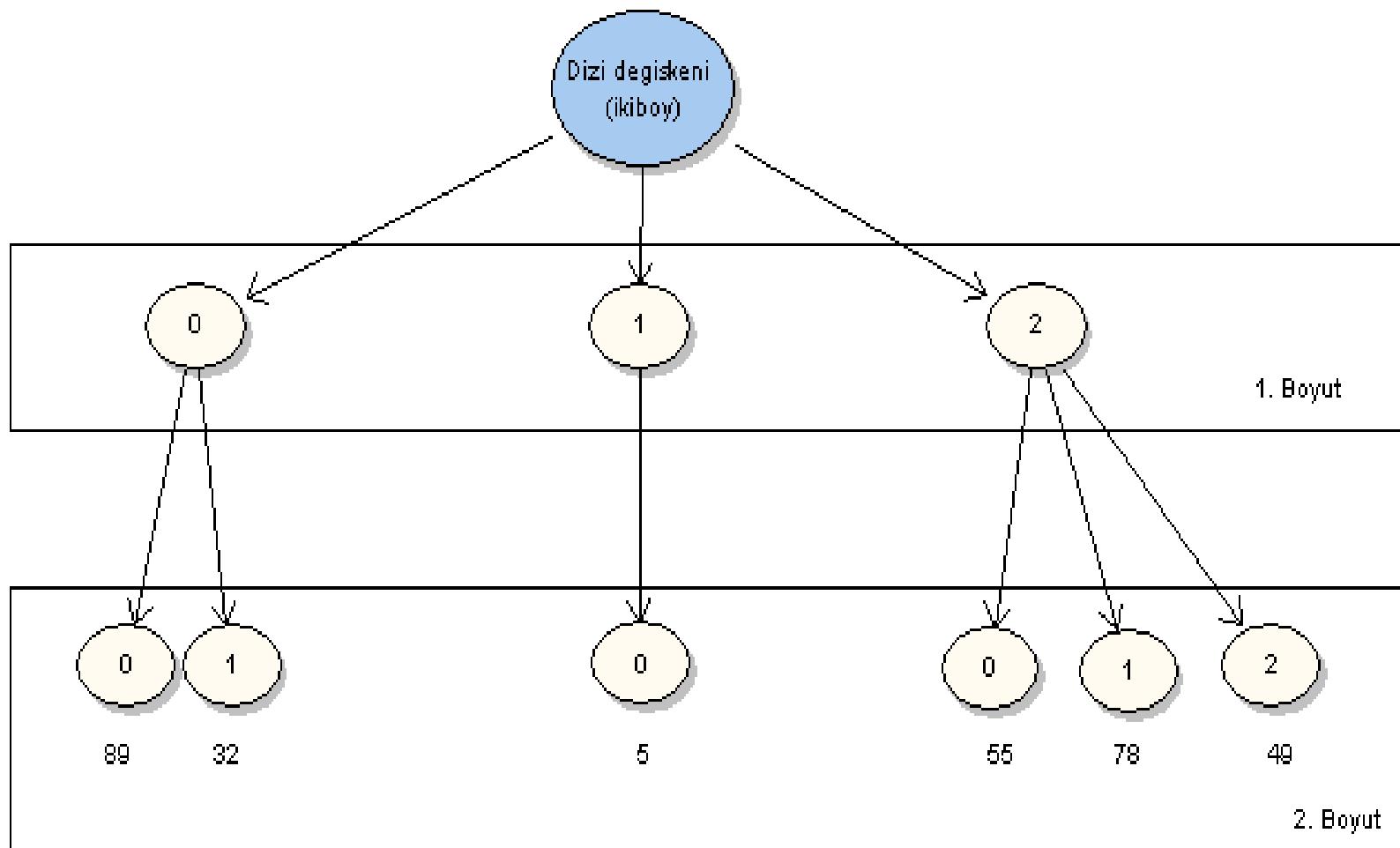
CokBoyutluDizilerOrnekBir.java



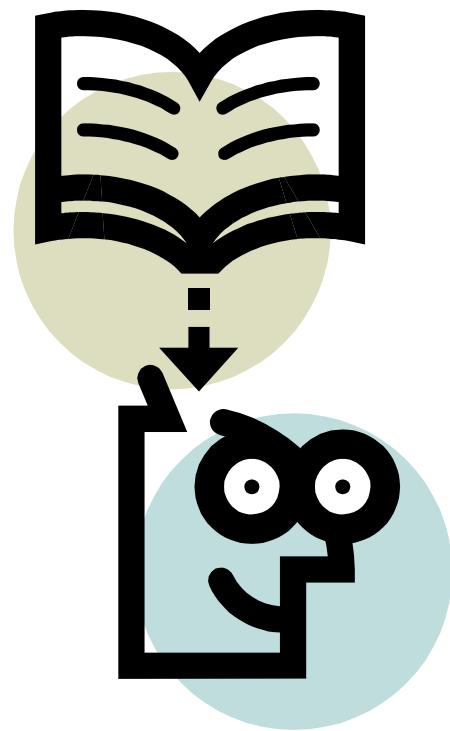
Değişik boyuta sahip diziler



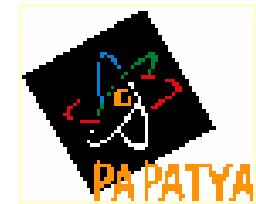
CokBoyutluDiziler.java



Sorular ...



Paket Erişimleri



- Erişim konusunda iki taraf vardır:
 - Kütüphaneyi kullanan kişiler (client)
 - Kütüphaneyi yazan kişiler

Paket (*package*)

- Paketler kütüphaneyi oluşturan elemanlardır.



PaketKullanim.java

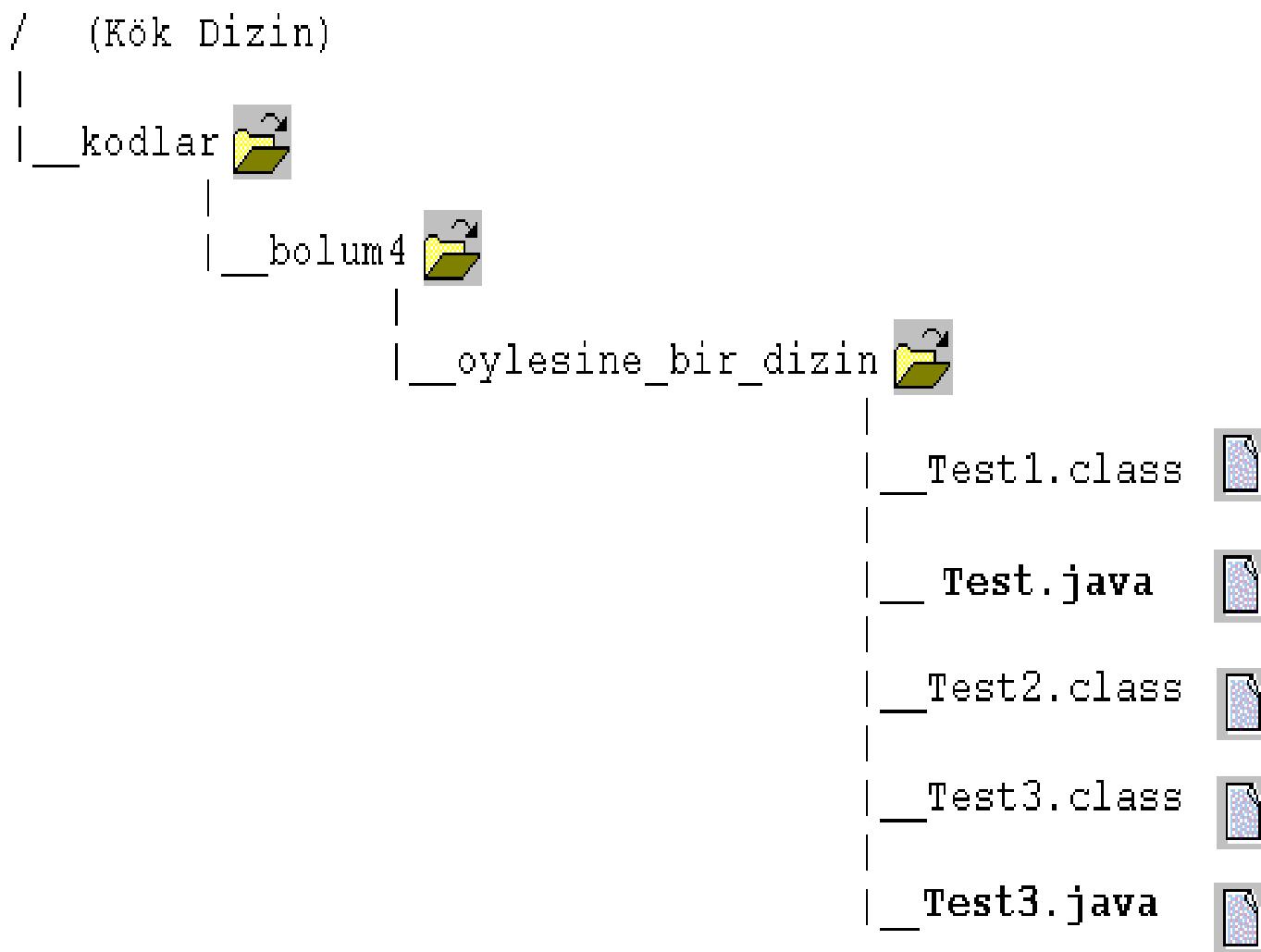
Varsayılan Paket (*Default Package*)



Test1.java



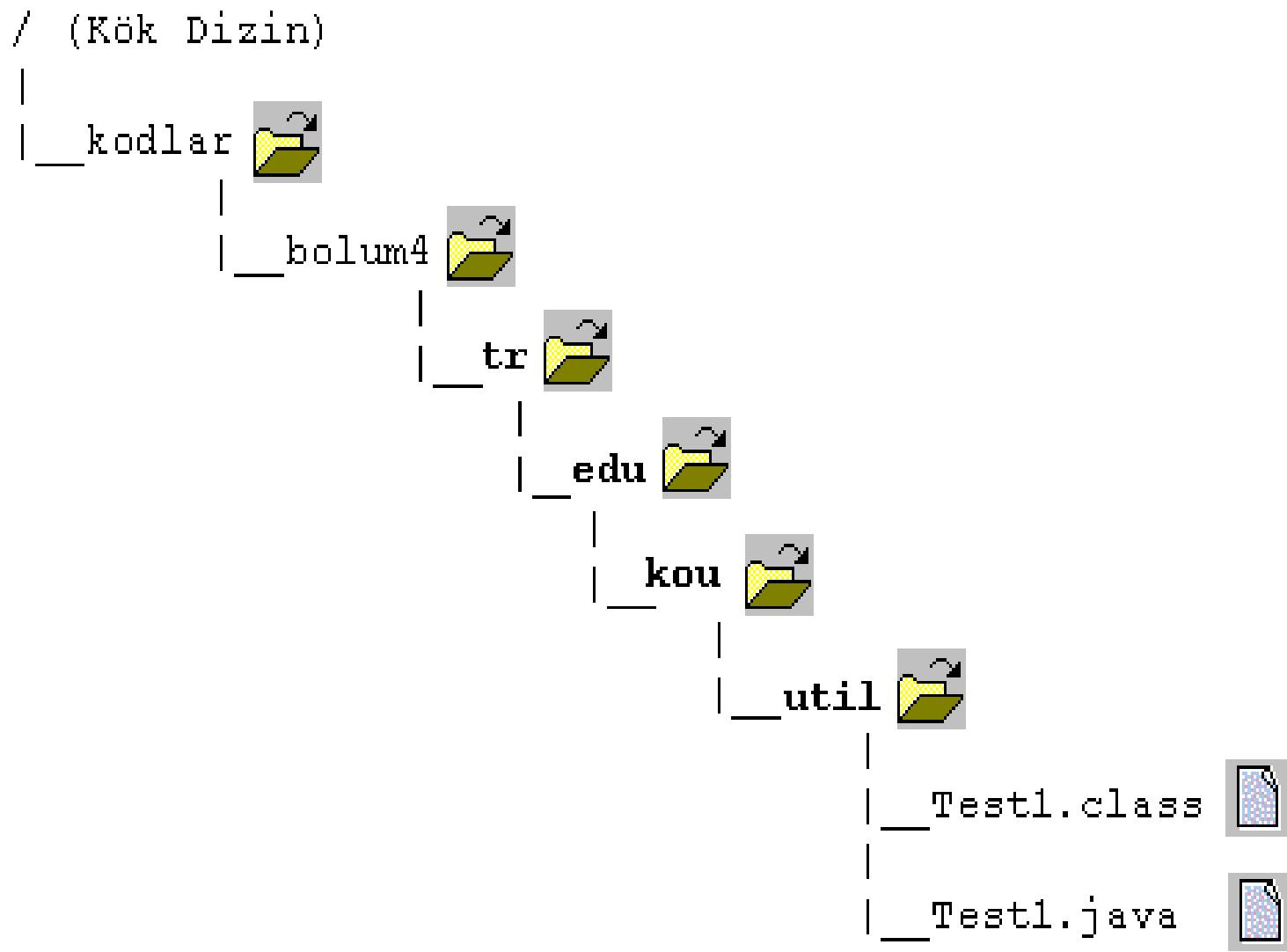
Test3.java

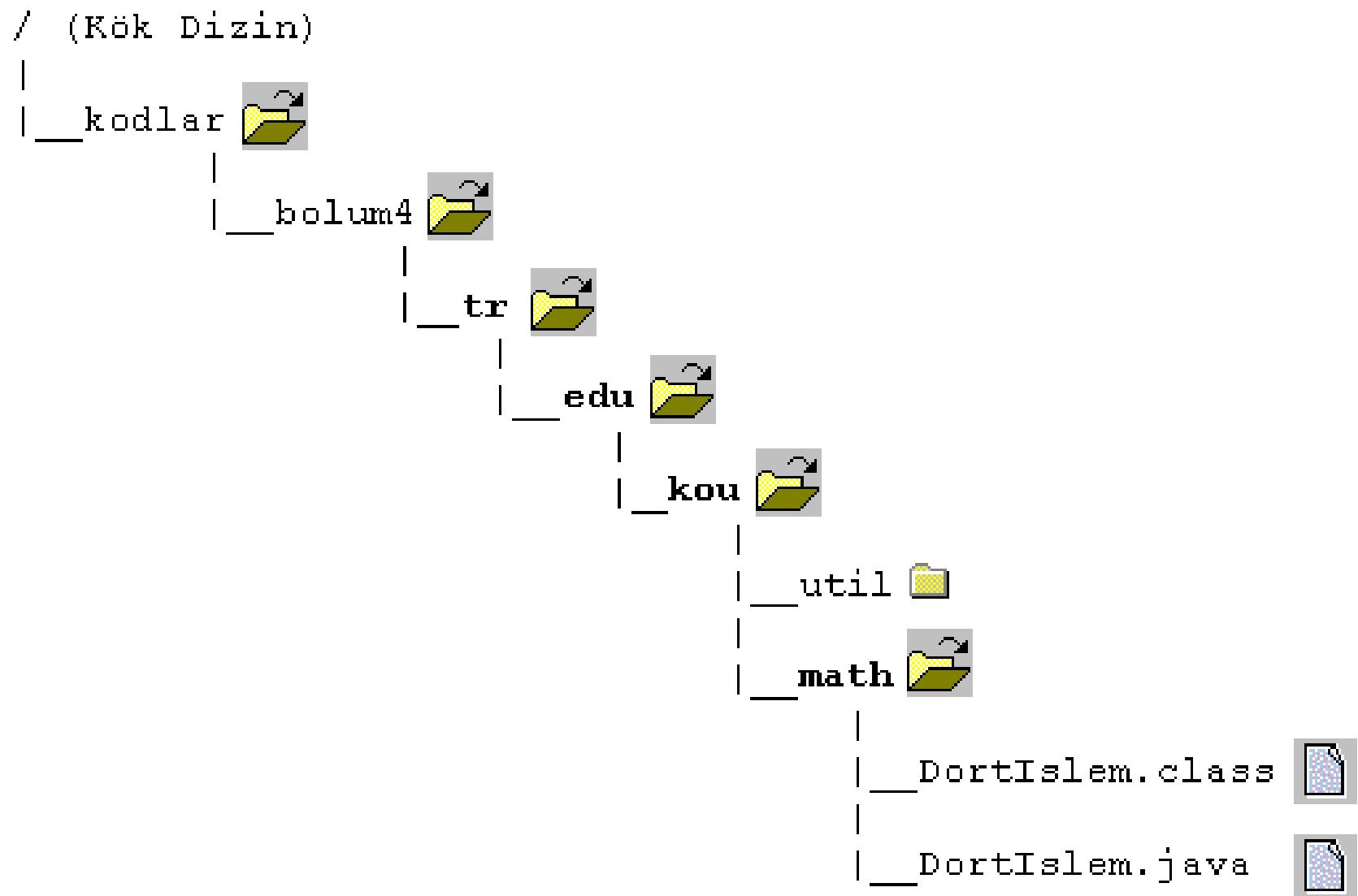


- Alan Adı Sistemi (Domain Name System)



tr/edu/kou/util/Test1.java



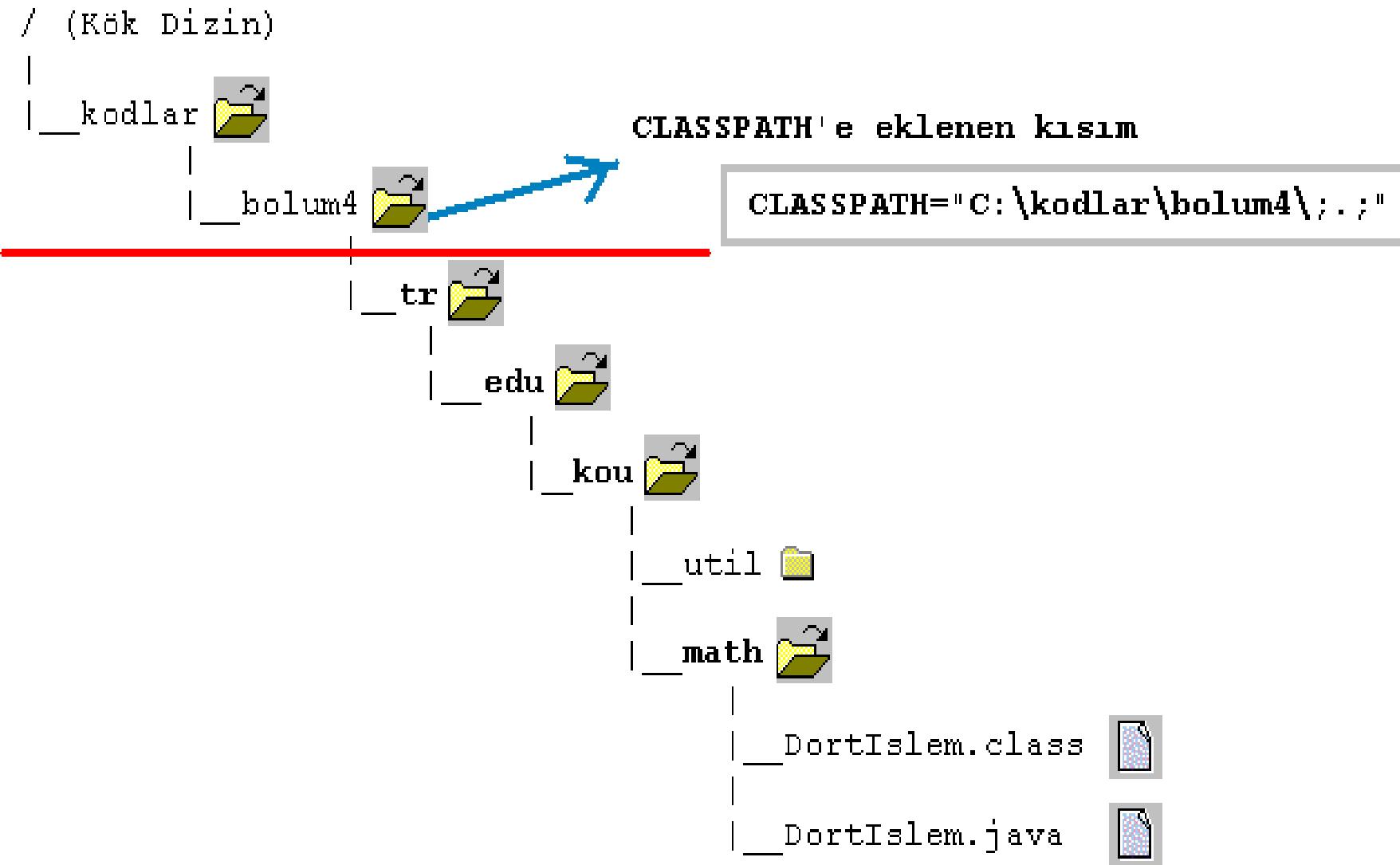


CLASSPATH Ayarları

```
C:\kodlar\bolum4\tr\edu\kou\math\
```

```
import tr.edu.kou.math.*;
```

```
CLASSPATH="C:\kodlar\bolum4\;."
```





tr/edu/kou/math/DortIslem.java

- Aşağıdaki örnek işletim sisteminin herhangi bir yerine yerleştirebilir.

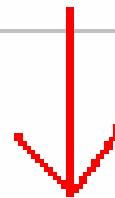


Hesaplama.java

Önemli nokta

- Varsayılan paketlerin birbirini görmesi için:

```
CLASSPATH="C:\kodlar\bolum4\;.;"
```



Dikkat !!

Çalışma



tr/edu/kou/util/ArrayList.java



Cakisma.java

- Çalışmayı önlemek için



Cakisma2.java

Paket içerisindeki uygulamaları çalıştırılmak



tr/edu/kou/math/Hesaplama.java

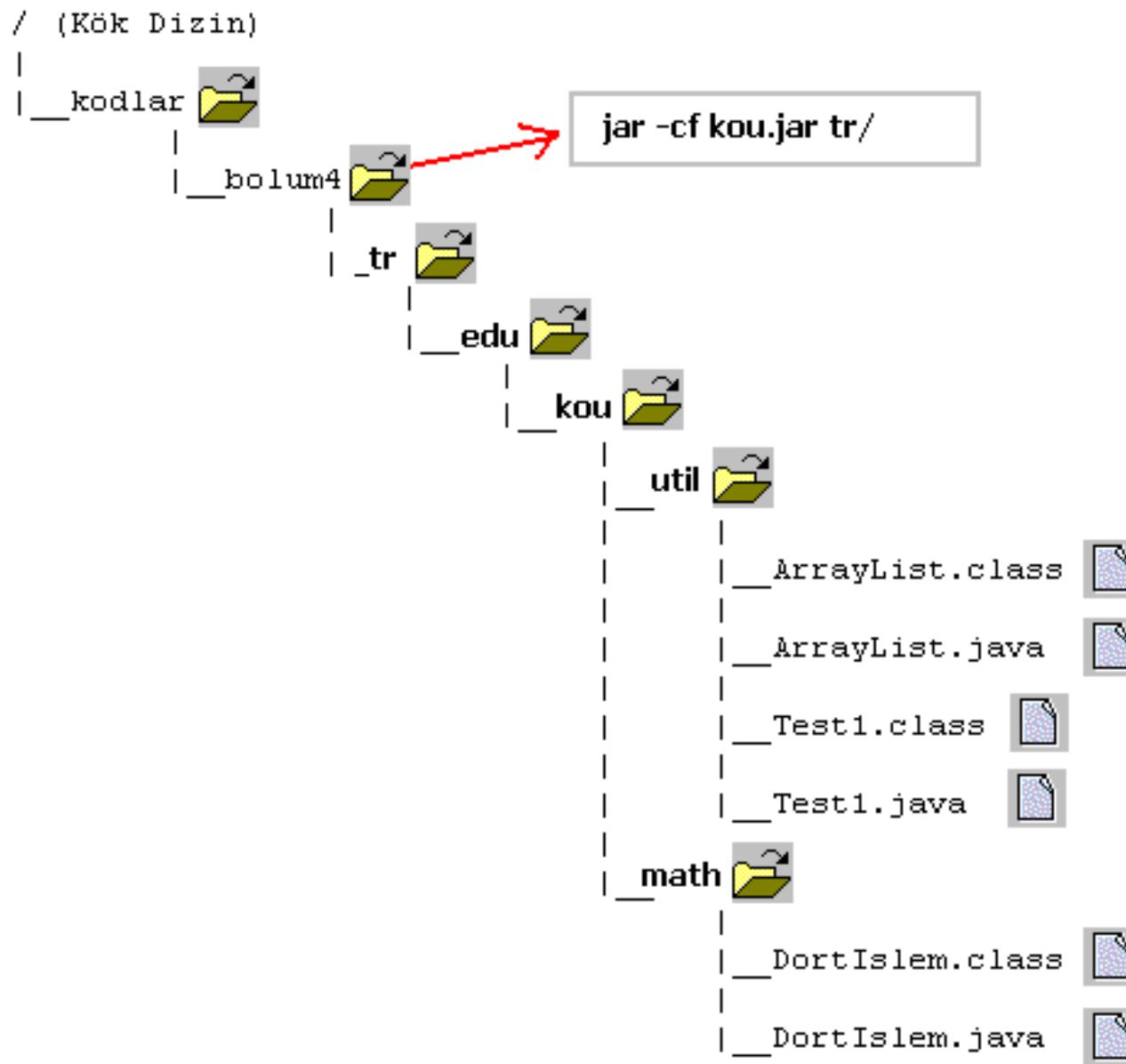
- Çalıştırma İşlemleri
 - `java Hesaplama`
 - `java tr.edu.kou.math.Hesaplama`

JAR Dosyaları (*The Java™ Archive File*)

- Güvenlik
- Sıkıştırma
- İndirme (*download*) zamanını azaltması
- Paket mühürleme(versiyon 1.2)
 - Versiyon uyumluluğu açısından
- Paket versiyonlanması (versiyon1.2)
 - Versiyon bilgilerinin gösterimi
- Taşınabilirlik

Açıklama	Komut
JAR dosyası oluşturmak için	jar -cf jar-dosya-ismi içeriye-atılacak-dosya(lar)
JAR dosyasının içeriğini bilmek için	jar -tf jar-dosya-ismi
JAR dosyasının içeriğini toptan dışarı çıkartmak için	jar -xf jar-dosya-ismi
Belli bir dosyayı JAR dosyasından dışarı çıkartmak için	jar -xf jar-dosya-ismi arşivlenmiş dosya(lar)
JAR olarak paketlenmiş uygulayı çalıştırmak için	jre -cp jar-dosya-ismi MainClass

```
jar -cf kou.jar tr/
```



CLASSPATH AYARLARI

CLASSPATH="C:\kodlar\bolum4\; . "

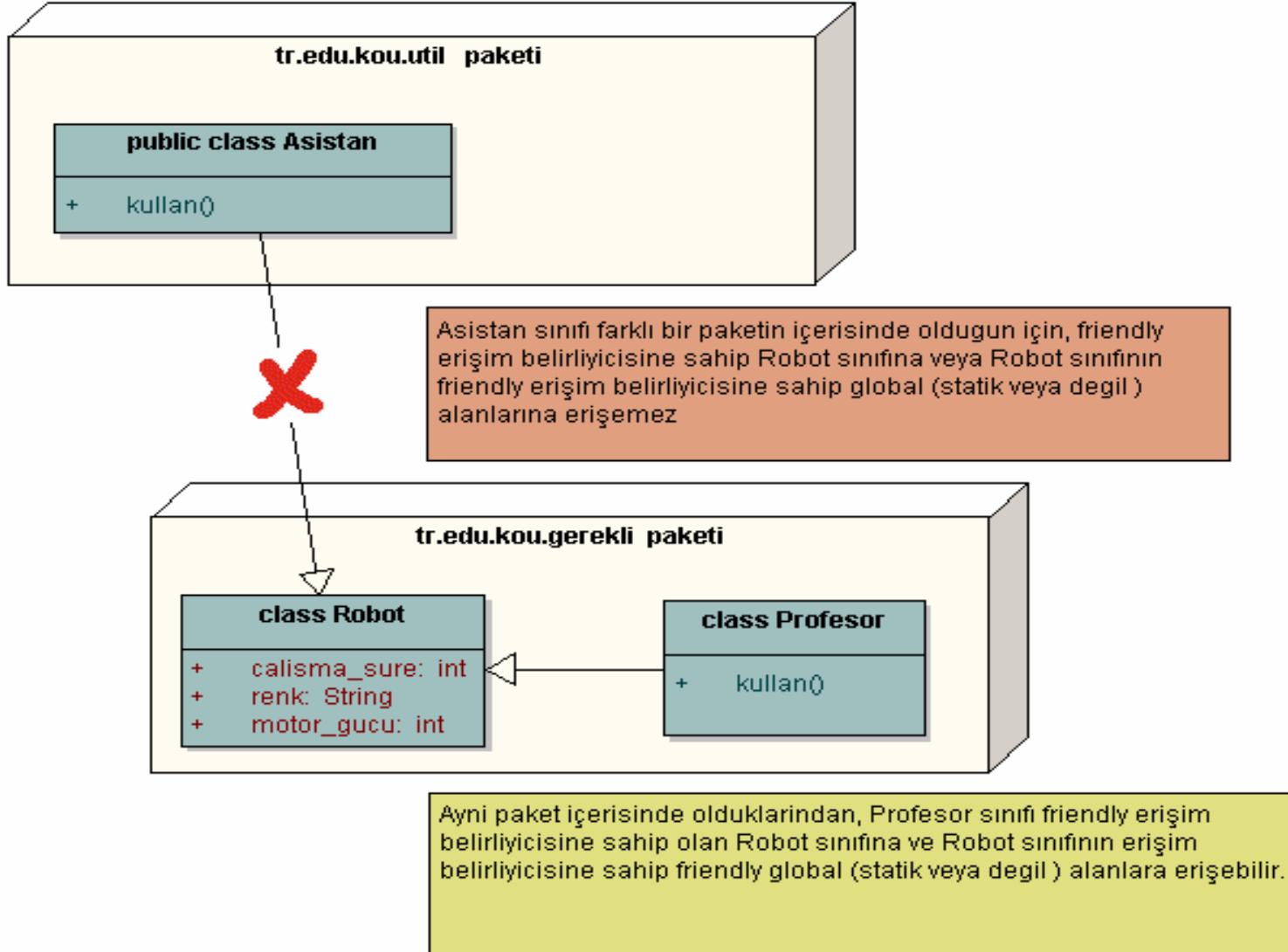
Yukarıdaki ifade yerine artık aşağıdaki ifadeyi kullanabiliriz.

CLASSPATH="C:\muzik\kou.jar; . "

- **friendly**
- **public**
- **protected**
- **private**

- Erişim belirleyiciler tüm global alanlar ve yordamlar için kullanılabilir.
- Global Alanlar
 - Statik veya değil
- Yordamlar (*methods*)
 - Statik veya değil

- Aynı paket içerisinde erişilebilmeyi sağlar.
 - Sınıflar
 - Global alanlar
 - Yordamlar (*methods*)friendly erişim belirleyicisine sahip olabilirler.





tr/edu/kou/gerekli/Robot.java



tr/edu/kou/gerekli/Profesor.java

- Başka bir paket



tr/edu/kou/util/Asistan.java

Varsayılan Paketlerde (*Default Package*) Erişim

```
class AltKomsu {  
    public static void main(String[] args) {  
        UstKomsu uk = new UstKomsu();  
        uk.merhaba();  
    }  
}
```

```
class UstKomsu {  
    void merhaba() {  
        System.out.println("Merhaba");  
    }  
}
```

```
public
```

- Heryerden erişilebilmeyi sağlar.
 - Sınıflar
 - Global alanlar
 - Yordamlar (*methods*)

public erişim belirleyicisine sahip olabilirler



tr/edu/kou/util/Makine.java



Makine sınıfını kullanan ***UstaBasi.java***

private (Özel)

- private erişim belirleyicisine sahip olan
 - Global değişkenler
 - Yordamlar (*methods*)

dışarıdan erişilemezler.
- **Sınıflar private olamazlar...**

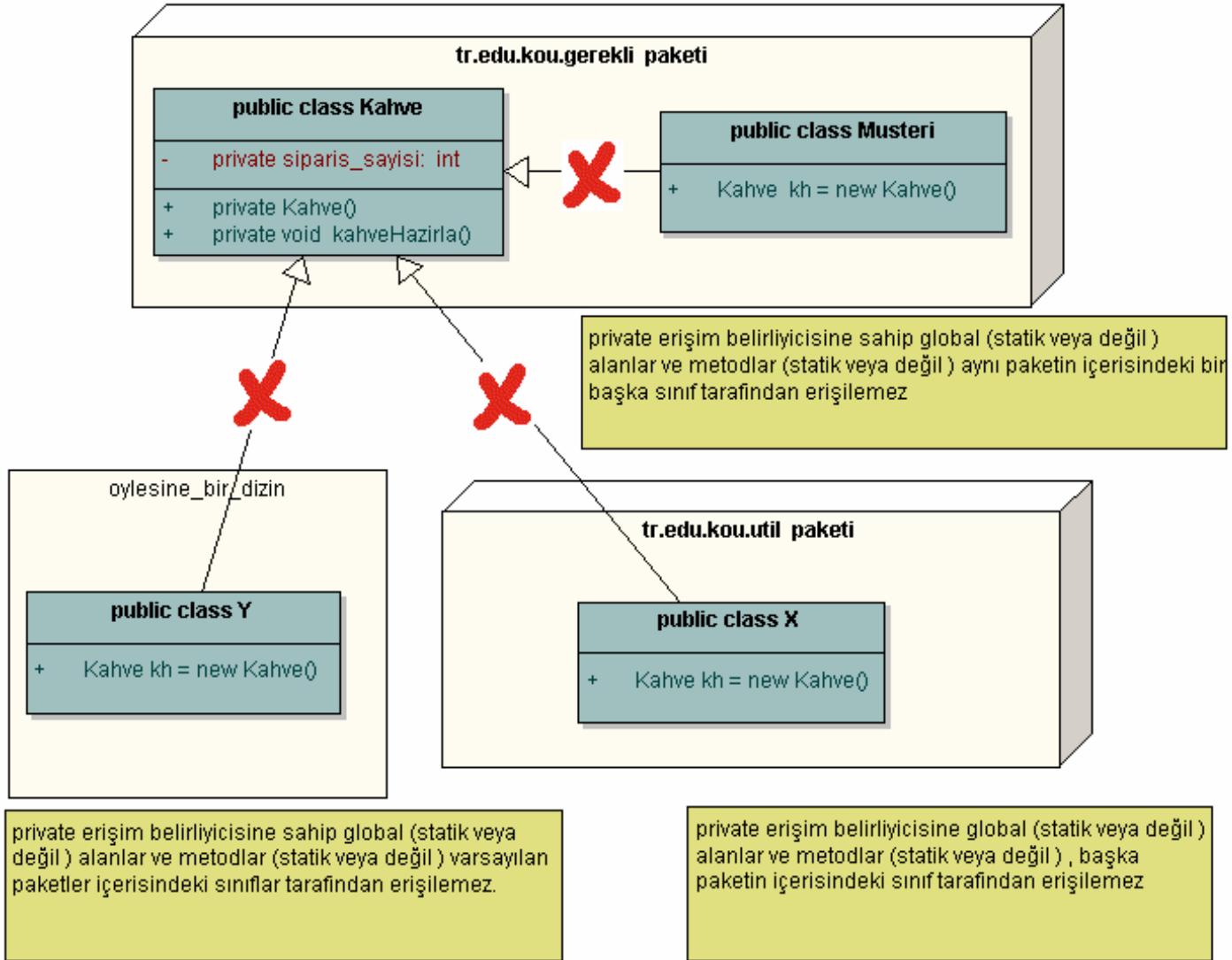
- Aynı paket içersindeki iki sınıf



tr/edu/kou/gerekli/Kahve.java



tr/edu/kou/gerekli/Musteri.java



protected

- protected erişim belirleyicisine sahip olan
 - Global alanlar
 - Yordamlar (*methods*)
- Erişim ancak bu sınıfın türeyen sınıflar ve aynı paket içerisindeki sınıflar tarafından erişilebilir.
- **Sınıflar protected olamaz.**

Kalıtım (Inheritance)

```
class Kedi extends Hayvan {  
    . . . . .  
    . . . .  
}
```

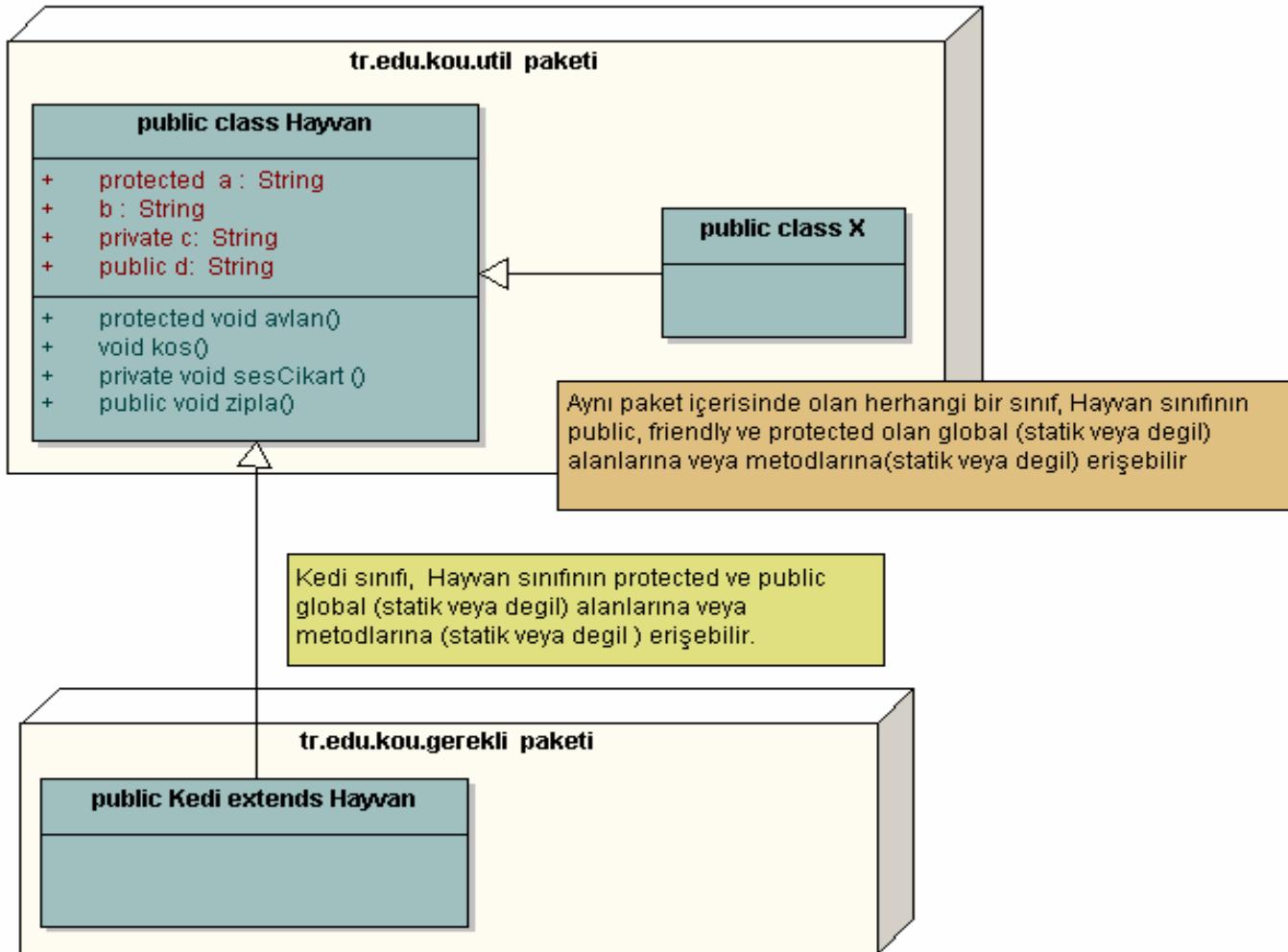
Örnek



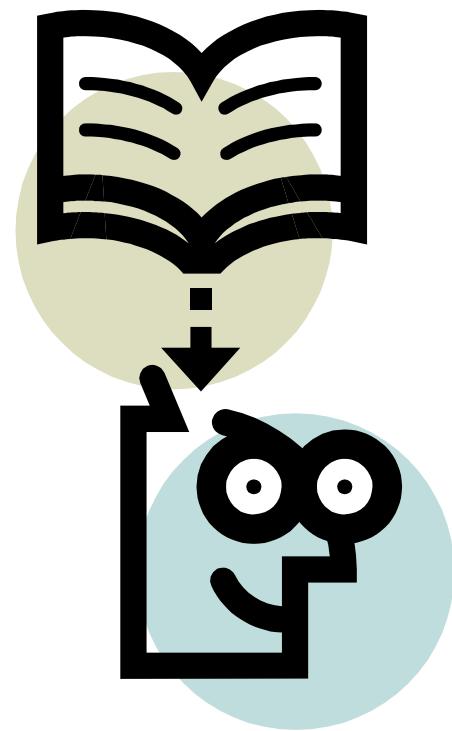
tr/edu/kou/util/Hayvan.java



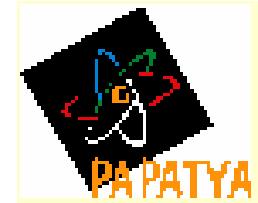
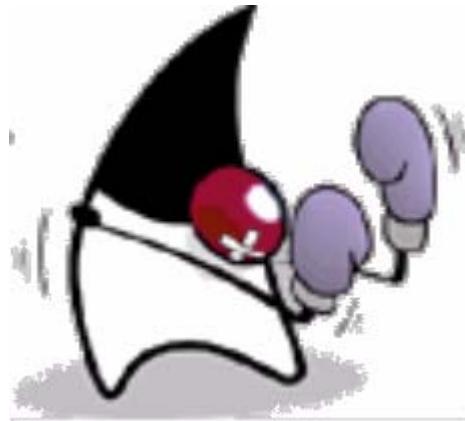
tr/edu/kou/gerekli/Kedi.java



Sorular ...



Sınıfların Tekrardan Kullanılması



İki Yöntem

- **Komposizyon (*Composition*)**
- **Kalıtım (*Inheritance*)**

Komposizyon(Composition)

```
class Meyva {  
    //...  
}
```

```
class Elma {  
    private Meyva m = new Meyva();  
    //...  
}
```

Komposizyon - UML



Örnek

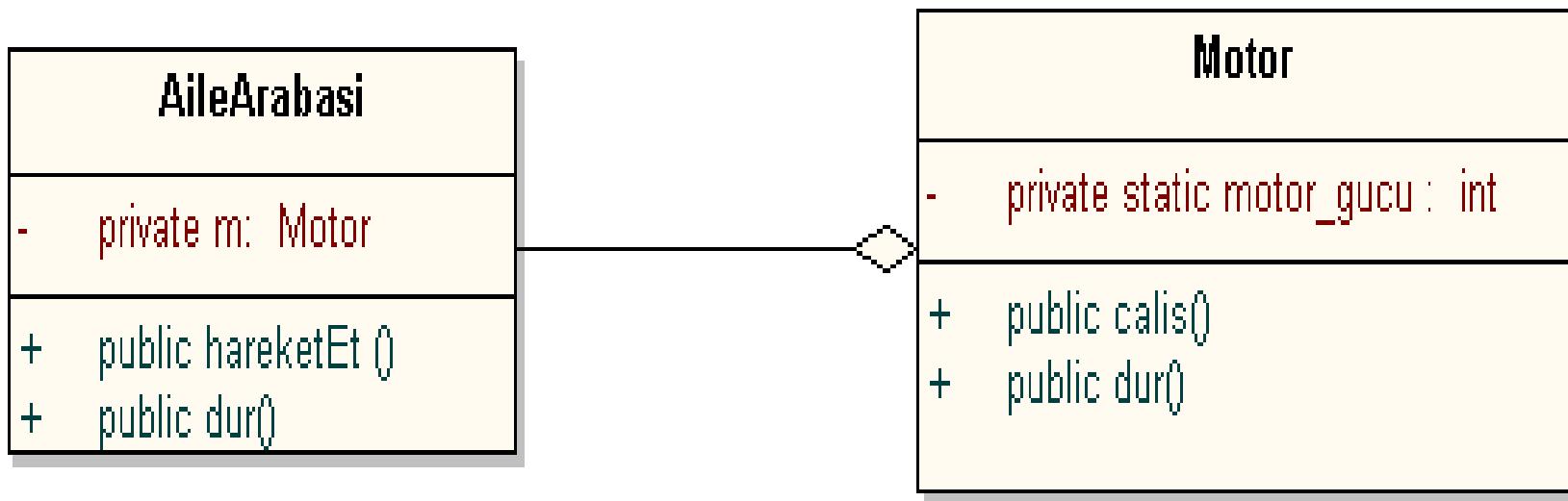


Motor.java

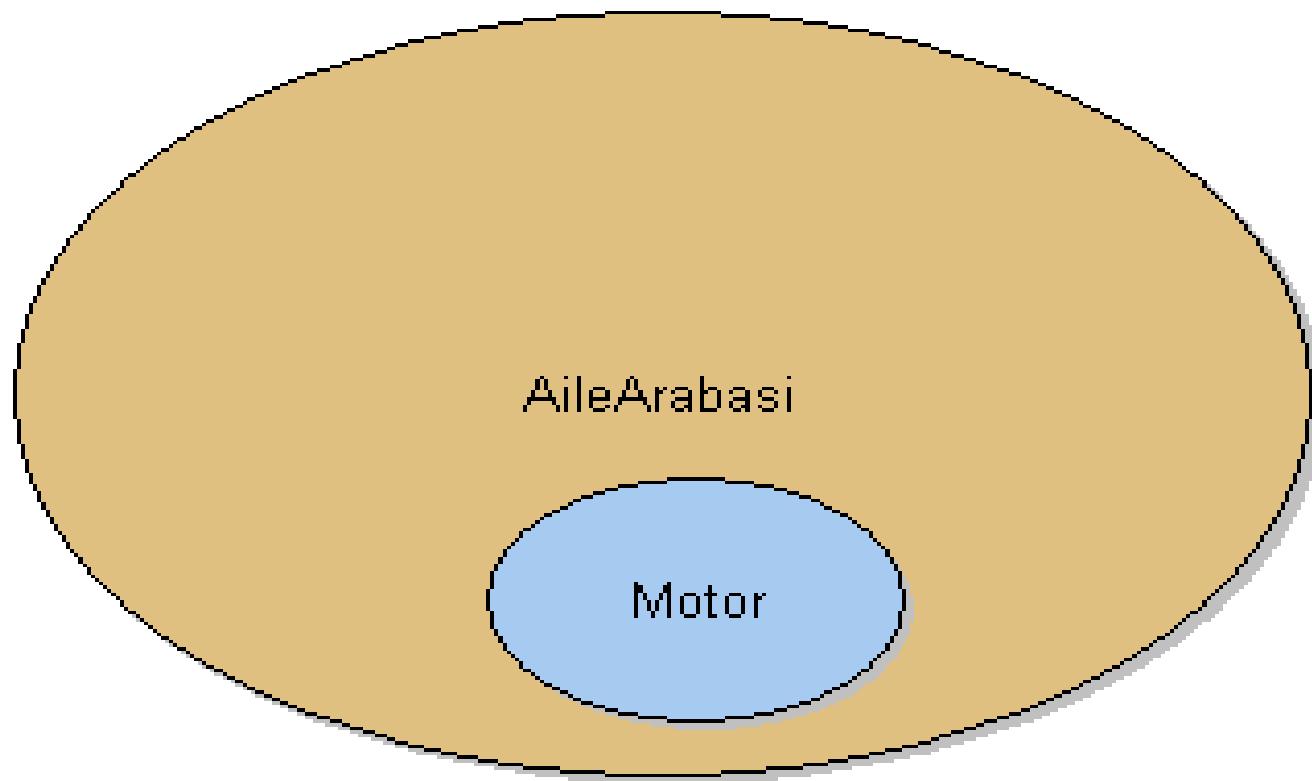


AileArabasi.java

Şekil



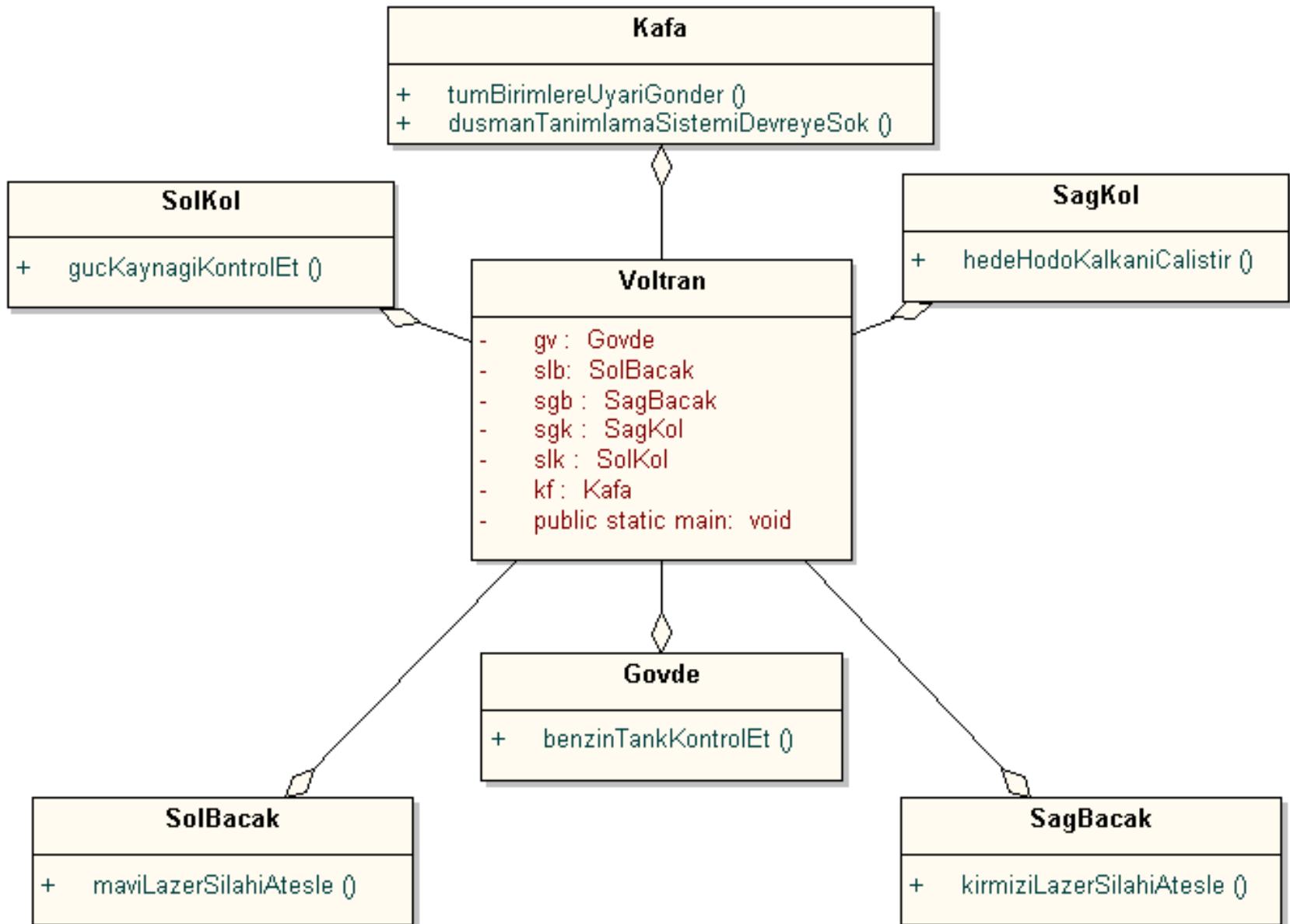
Kuş Bakışı



Örnek



Voltran.java

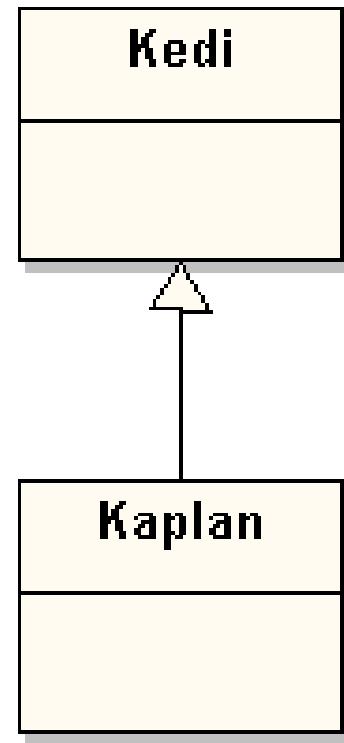


Kalıtım (*Inheritance*)

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

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

Kalıtım - UML



Örnek



KediKaplan.java



YeniBirSinif.java

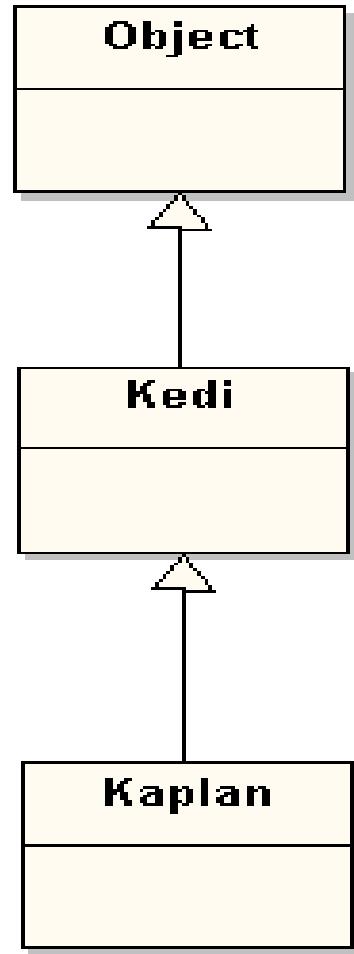
Gizli Nokta

```
public class YeniBirSinif extends Object {
```

Nesne yordamları (*methods*)

- **clone()** : Bu nesnenin aynısını klonlar.
- **equals(Object obj)** : obj nesnesi , bu nesneye eşit mi kontrolü yapar.
- **finalize()** : İlgili nesne bellekten silinmeden hemen önce çağrılan yordam.
- **getClass()** : Bu nesnenin çalışma anında sınıf bilgilerini geri döner .
- **hashCode()** : Bu nesnenin hash kodunu geri döner .
- **notify()** : Bu nesnenin bekleme havuzunda olan tek iş parçacığını (*thread*) uyandırır. (ilerleyen bölümlerde inceleyeceğiz)
- **notifyAll()** : Bu nesnenin bekleme havuzundaki tüm iş parçacıklarını uyandırır. (ilerleyen bölümlerde inceleyeceğiz)
- **toString()** : Bu nesnenin String tipinden ifadesini geri döner .
- **wait()** : O andaki iş parçacığının (*thread*) beklemesini sağlar; bu bekleme notify() veya notifyAll () yordamları sayesinde sona erer.
- **wait (long timeout)** : O andaki iş parçacığının belirtilen süre kadar beklemesini sağlar; bu bekleme notify() veya notifyAll () yordamları sayesinde de sona erebilir.
- **wait (long timeout , int nanos)** : O andaki iş parçacığının belirtilen gerçek süre kadar beklemesini sağlar ; bu bekleme notify() veya notifyAll () yordamları sayesinde de sona erebilir.

Kalıtım-UML

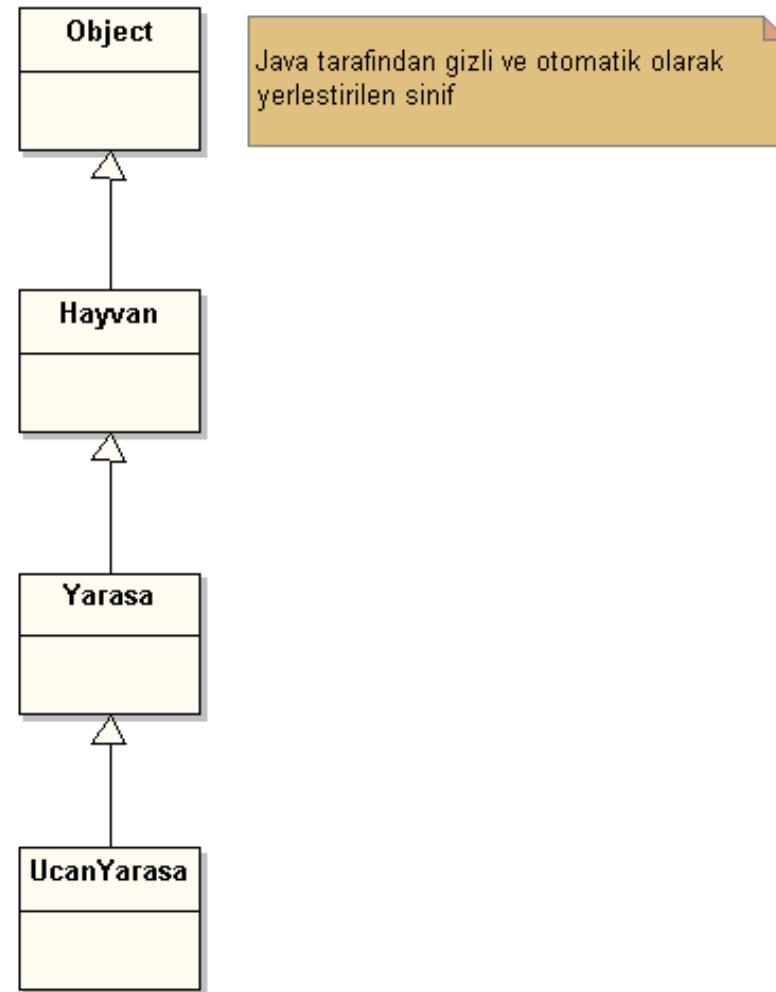


Kalıtım ve ilk değer alma sırası



IlkDegerVermeSirasi.java

Kalıtım-UML



Paremetre alan yapılandırmaları ve kalıtım



IlkDegerVermeSirasiParametreli.java

İlkDegerVermeSırasıParametreli.java-hatalı

```
class Insan {  
    public Insan(int par) {  
        System.out.println("Insan Yapilandiricisi " + par);  
    }  
}  
class ZekiInsan extends Insan {  
    public ZekiInsan(int par) {  
        System.out.println("ZekiInsan Yapilandiricisi " + par);  
        super(par+1); // ! hatalı !  
    }  
}  
class Hacker extends ZekiInsan{  
    public Hacker(int par) {  
        System.out.println("Hacker Yapilandiricisi " + par);  
        super(par+1); // ! hatalı !  
    }  
    public static void main(String args[]) {  
        Hacker hck = new Hacker(5);  
    }  
}
```

Kompozisyon mu ? Kalıtım mı ?

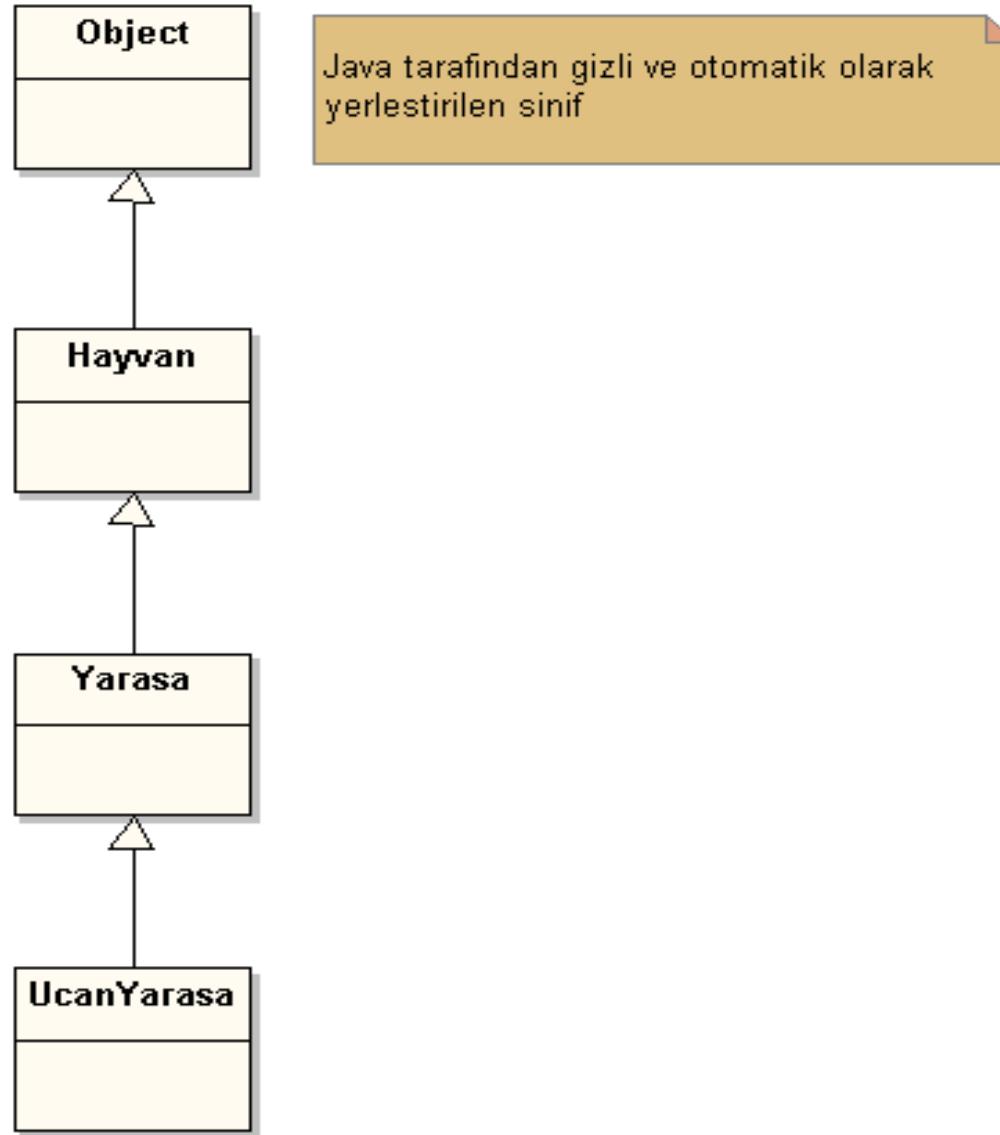
- Hangi yöntemi ne zaman tercih etmeliyiz ?
 - Kompozisyon(*Composition*)
 - Kalıtım (*Inheritance*)

Komposizyon



Araba.java

- Sınıflar arasında **bir** ilişkisi olmalıdır.
 - UçanYarasa **bir** Yarasadır.
 - Yarasa **bir** Hayvandır .
 - O zaman UçanYarasa da **bir** Hayvandır.
 - Hayvan da **bir** Nesnedir.



İptal etmek (*Overriding*)

- Türemiş sınıfların, ana sınıflar içerisindeki yordamları (*methods*) iptal edilebilir.



KitapEvi.java (düz)



KitapEvi2.java (iptal eden)

Erişim Belirleyiciler ve İptal etme (Override)

- *public*
- *protected*
- *friendly*
- *private*



Telefonlar.java (Yanlış)



Hesap.java (Doğru)

İptal etmek(*Overriding*) ve adaş yordamlarının(*Overload*) birbirlerine karıştırılması

- Bu iki kavram birbirlerine kolaylıkla karıştırılabilir.



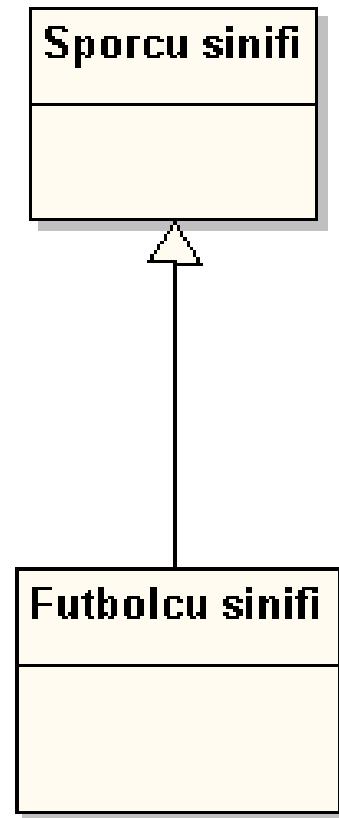
CalisanMudur.java

Yukarı Çevirim (*Upcasting*)



Spor.java

Yukarı Çevirim (*Upcasting*)-UML



Final Kavramı

- Final kelimesinin sözlük anlamı "son" demektir .
- Java programlama dilinde **final** anahtar kelimesi değiştirilemezliği simgeler.

Global Alanlar ve Final Kavramı

- Derleme anında değerlerini bilebildiğimiz **final** global alanlar.
- Çalışma anında değerlerini bilebildiğimiz **final** global alanlar.

Örnek



FinalOrnek.java

Final parametreler



FinalParametre.java

Boş (*Blank*) Final



BosFinal.java

Final Yordamlar

- **final** yordamlar türetilmiş sınıflar tarafından iptal edilemezler (*override*).



FinalMetod.java

private ve final

- **final** ve **private** erişim belirleyicisine sahip olan bir yordam, başka bir yordam tarafından iptal ediliyormuş gibi gözükebilir.



SivilPolis.java

Final Sınıflar

- **final** sınıflardan türetilme yapılamaz!



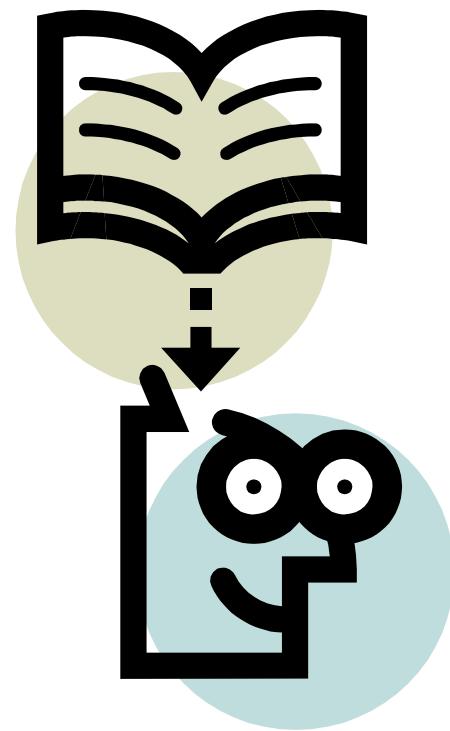
Tv.java

Kalıtım (*Inheritance*) ve ilk değer alma sırası

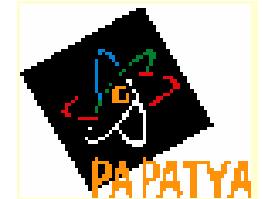
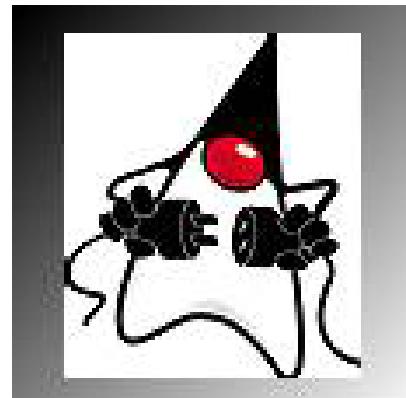


Bocekcik.java

Sorular ...



Polimorfizm



Polimorfizm

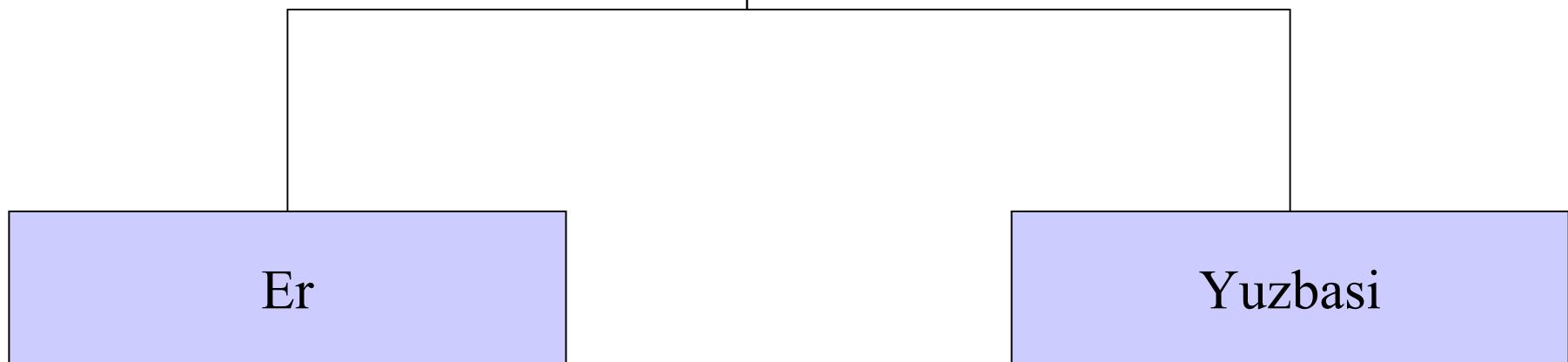
- Polimorfizm, nesneye yönelik programlamanın (OOP) önemli özelliklerinden biridir.
- Eski Yunanca bir sözcük olup "bir çok şekil" anlamına gelmektedir .
- Polimorfizm ile kalıtım konusu iç içedir.

Örnek



PolimorfizmOrnekBir.java

Asker



Polimorfizm

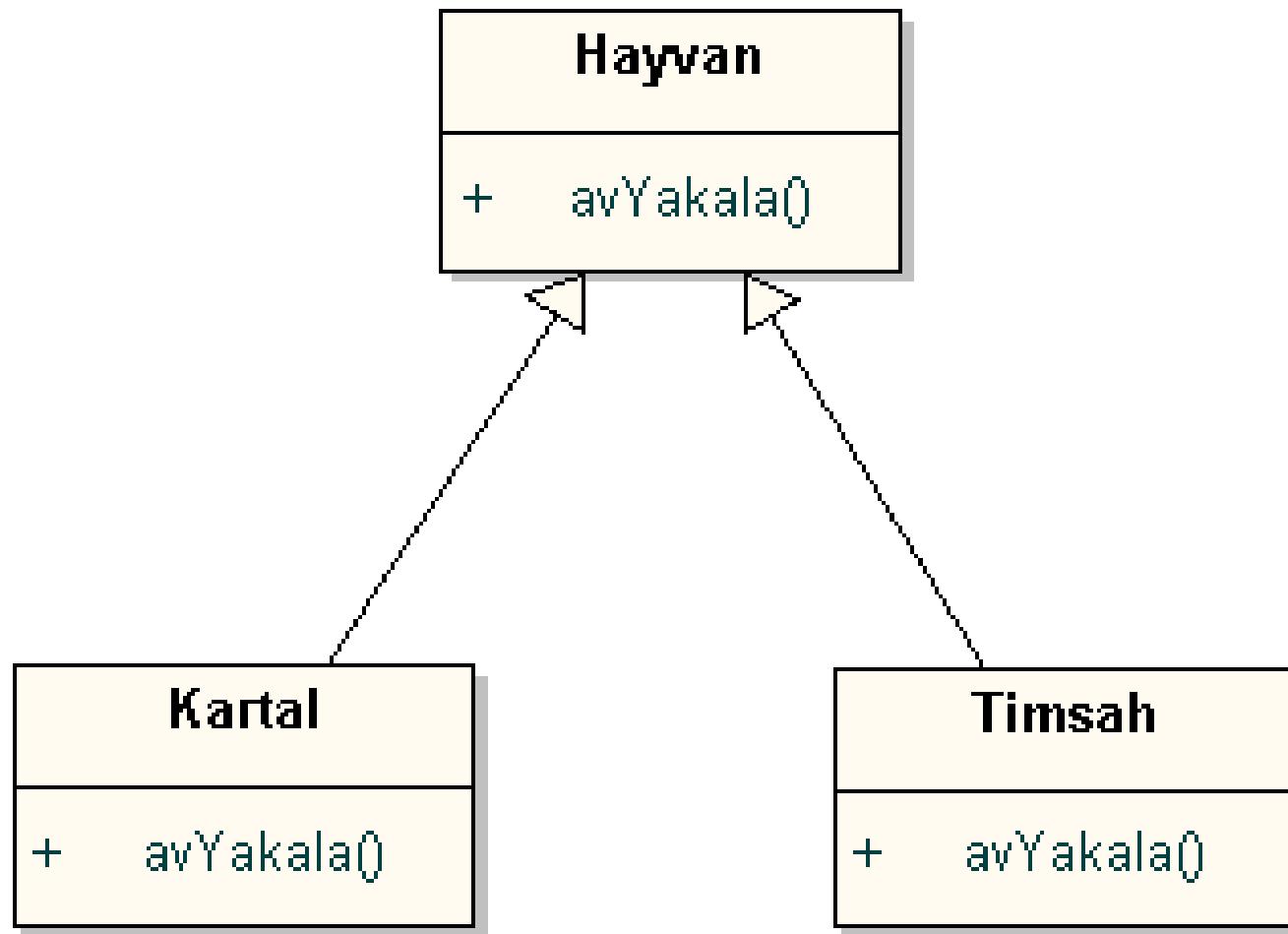
- Asker a = **new** Asker();
- Asker a = **new** Er();
- Asker a = **new** Yuzbasi();

Geç bağlama (late binding) - I

- Polimorfizm olmadan geç bağlama özelliğinden bahsedilemez.



PolimorfizmOrnekIki.java



java PolimorfizmIki

Kartal avYakala

Hayvan avYakala

Kartal avYakala

java PolimorfizmIki

Timsah avYakala

Timsah avYakala

Hayvan avYakala

java PolimorfizmIki

Timsah avYakala

Hayvan avYakala

Kartal avYakala

Geç bağlama (late binding) - II

- 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*).

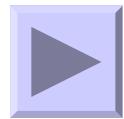
final ve Geç bağlama

- **final** özelliğinin kullanılmasının iki sebebi olabilir.
 - tasarım .
 - verimlilik .

Cevap

- Uygulama içerisinde herhangi bir nesneye ait normal bir yordam (final olmayan) çağrıldığında :
 - Java, acaba doğru nesnenin uygun yordamı mı çağrılıyor diye bir kontrol yapar.
- Daha doğrusu geç bağlamaya (*late-binding*) ihtiyaç var mı kontrolü yapılır.

Örnek



KediKaplan.java (on)



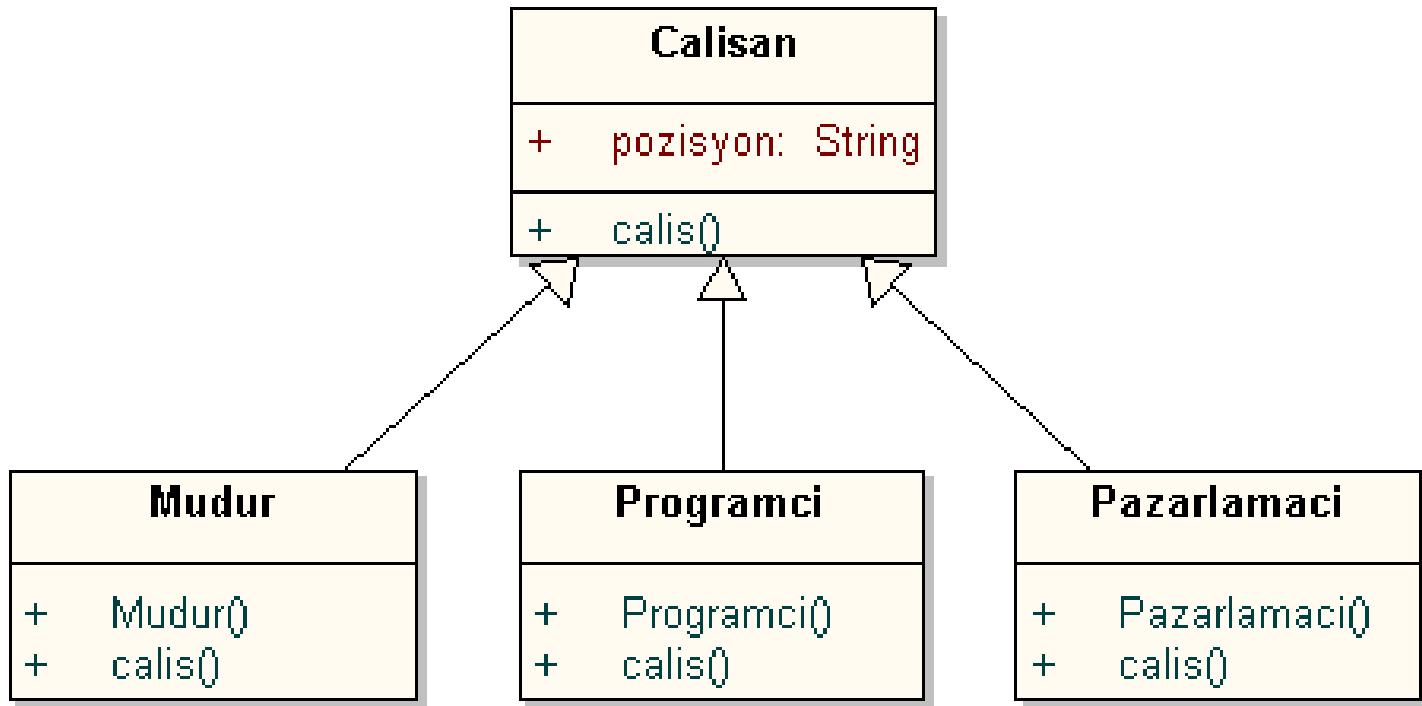
KediKaplan2.java (off)

Neden Polimorfizm ?

- Polimorfizm olmasaydı neler olurdu ?



IsYeriNon.java



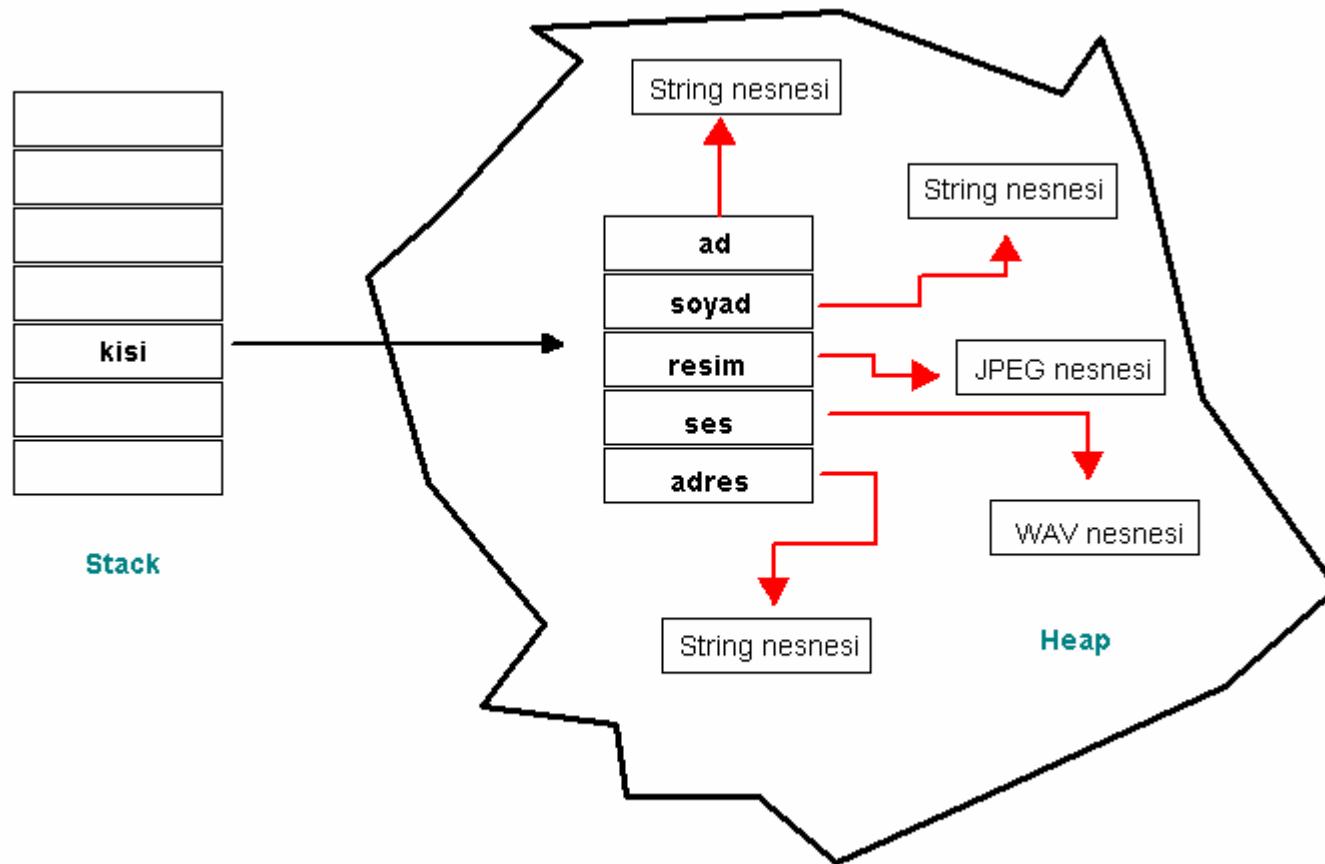
Neden Polimorfizm ?

- *IsYeriNon.java* örneğimizi nesneye yönelik programlama çerçevesinde tekrar yazarsak :



IsYeri.java

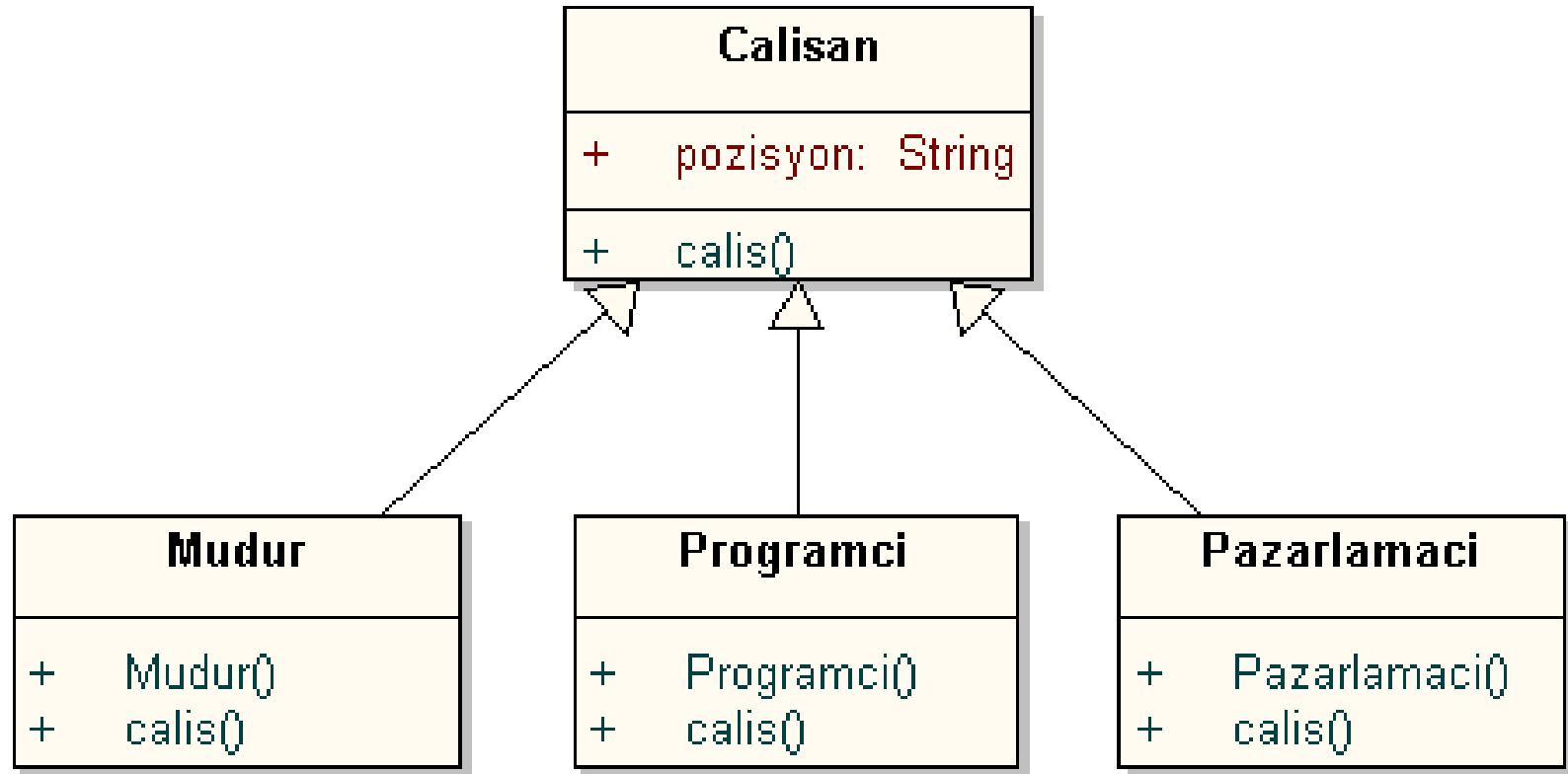
Nesnelerden oluşan bir dizi



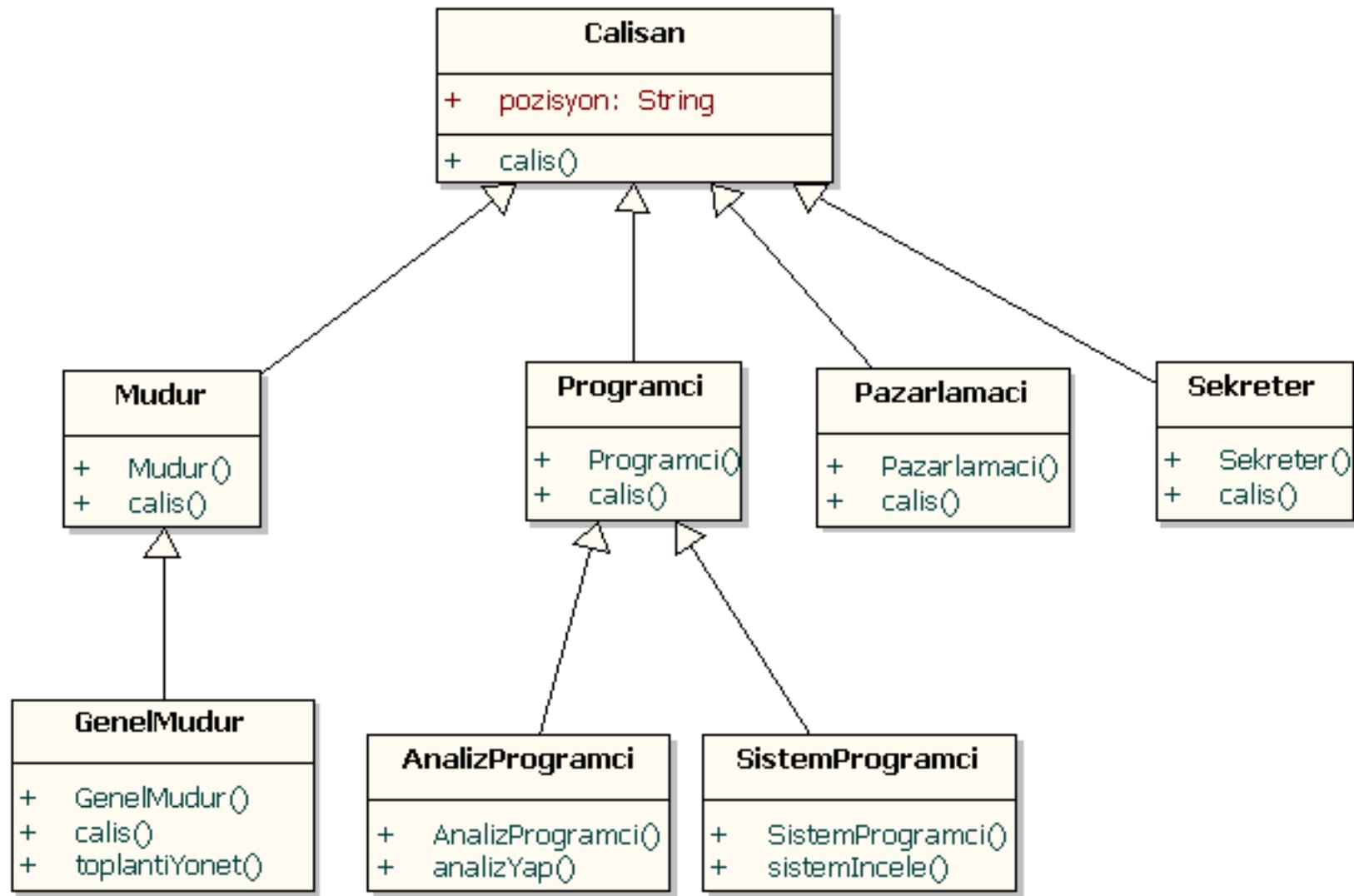
Genişletilebilirlik (Extensibility)

- Genişletilebilirlik, mevcut kalıtımsal hiyerarşiyi genişletmektir.
- Polimorfizm özelliği sayesinde genişletilebilirlik çok basitçe indirgenmiş bulunmaktadır .

Sınıf hiyerarşisi – IsYeri.java



Sınıf hiyerarşisi – BuyukIsYeri.java



Genişletilebilirlik (Extensibility)-2

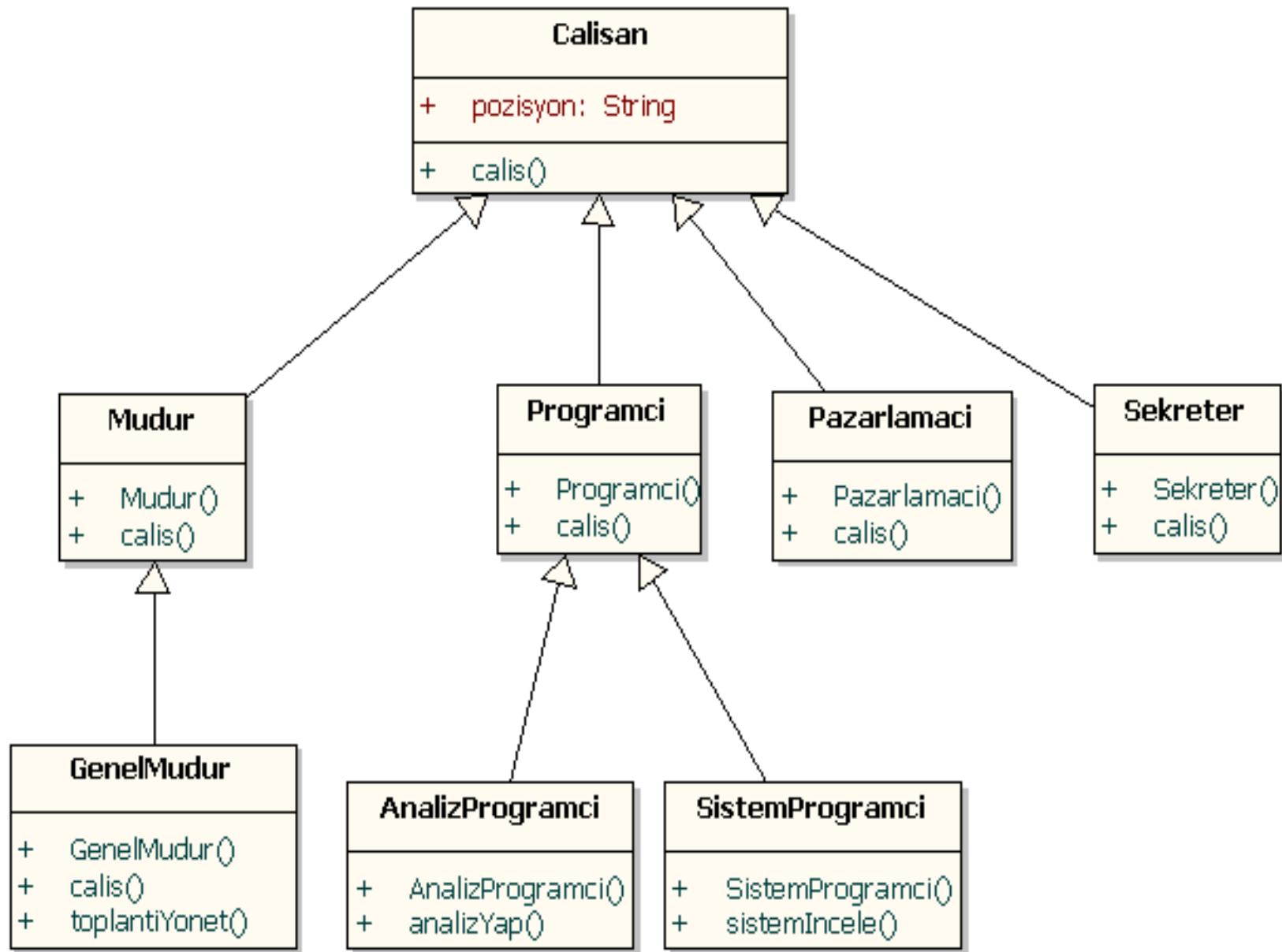
- Polimorfizm sayesinde, mevcut hiyerarşik yapı ne kadar genişletilirse genişletilsin, yordam (*method*) çağrıma yapısı hep aynı kalır.



BuyukIsyeri.java

Soyut sınıflar ve Yordamlarlar

(Abstract Classes and Methods)



Örnek



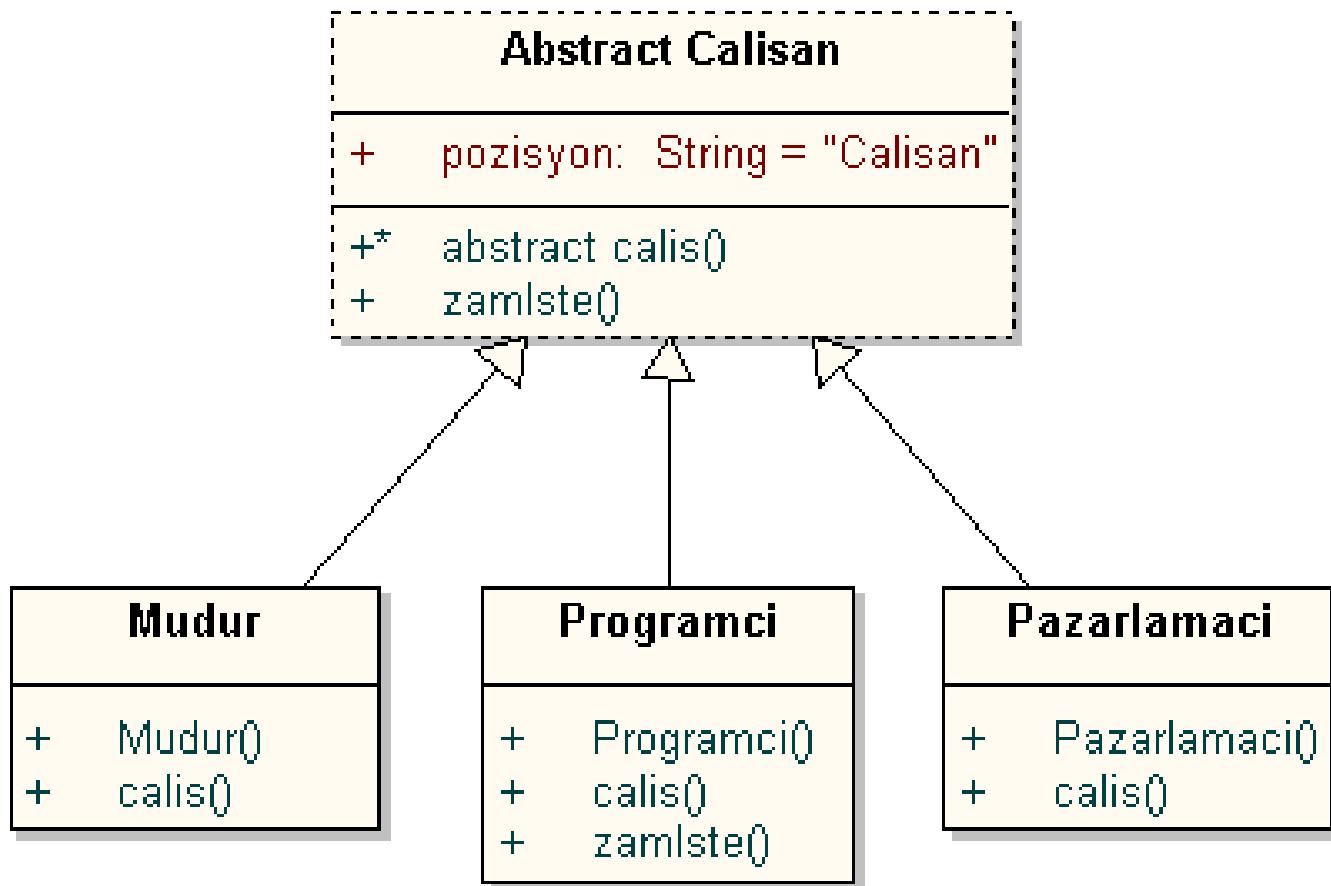
BuyukIsyeri.java

```
class Calisan {  
    public String pozisyon = "Calisan";  
    public void calis() {}  
}
```

Soyut Sınıflar (Abstract classes)

- Soyut sınıfların içerisinde en az bir tane gövdesiz (soyut) yordam bulunur.
- Soyut sınıfları direkt **new()** anahtar kelimesi ile oluşturamayız.

```
abstract void calis() ; // gövdesi olmayan soyut yordam
```



Örnek



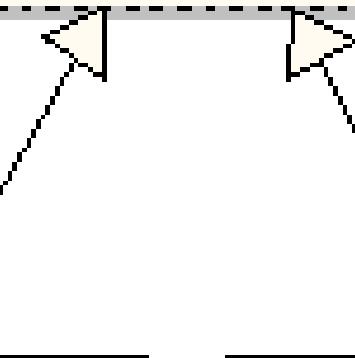
AbIsYeri.java

Niye soyut sınıf ve yordamlara ihtiyaç duyuyoruz ?

- Eğer bir işlem değişik verilere ihtiyaç duyup aynı işi yapıyorsa, bu işlem soyut (*abstract*) sınıfların içerisinde tanımlanmalıdır.

Abstract Cizim

```
+* abstract noktaCiz(int, int)  
+     cizgiCiz(int, int, int, int)
```



CepTelefonuCizim

```
+     noktaCiz(int, int)
```

MonitorCizim

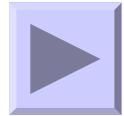
```
+     noktaCiz(int, int)
```

Örnek



CizimProgrami.java

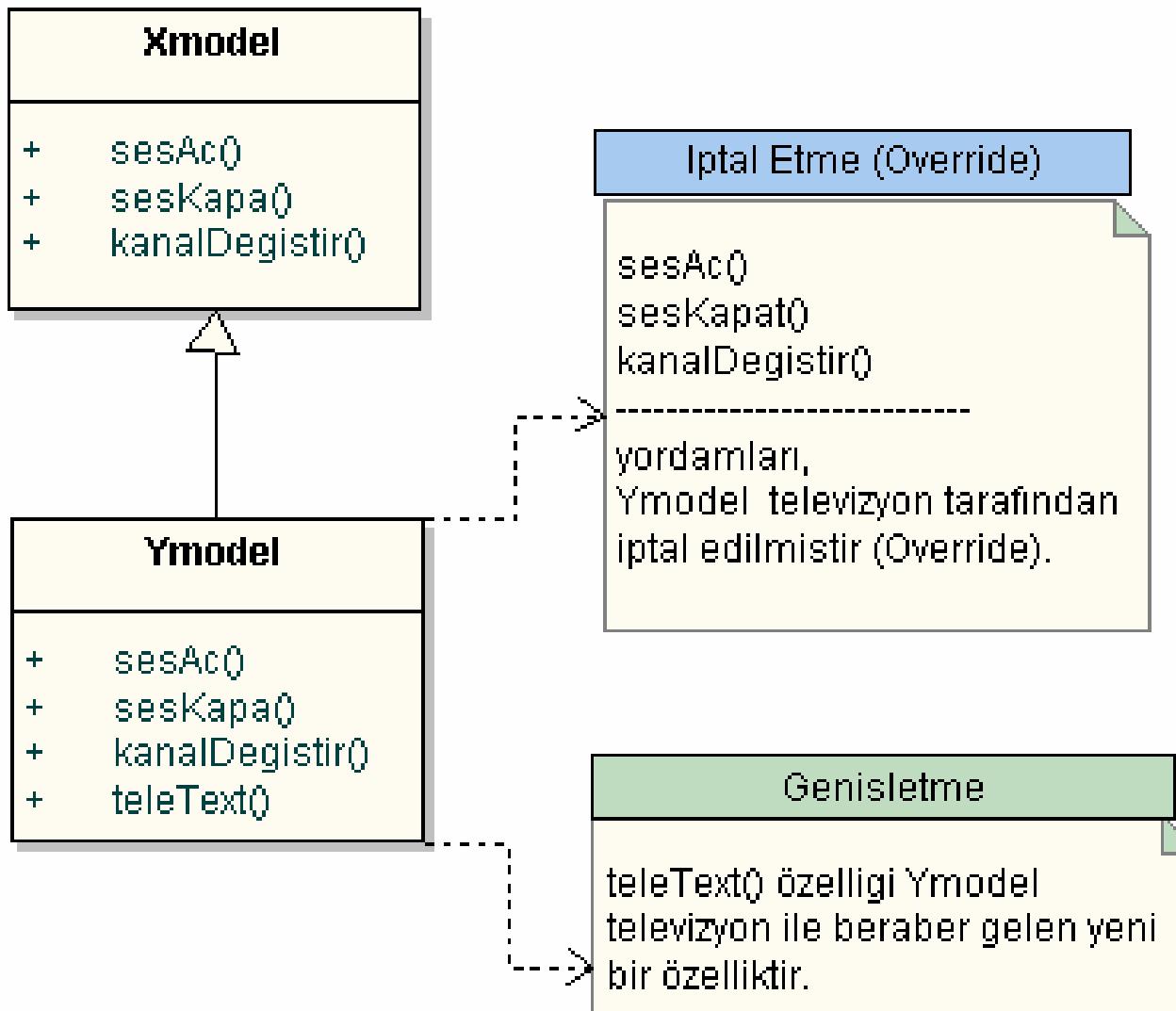
Yapılandırıcılar içerisindeki ilginç durumlar...



Spor.java

Ekran Çıktısı

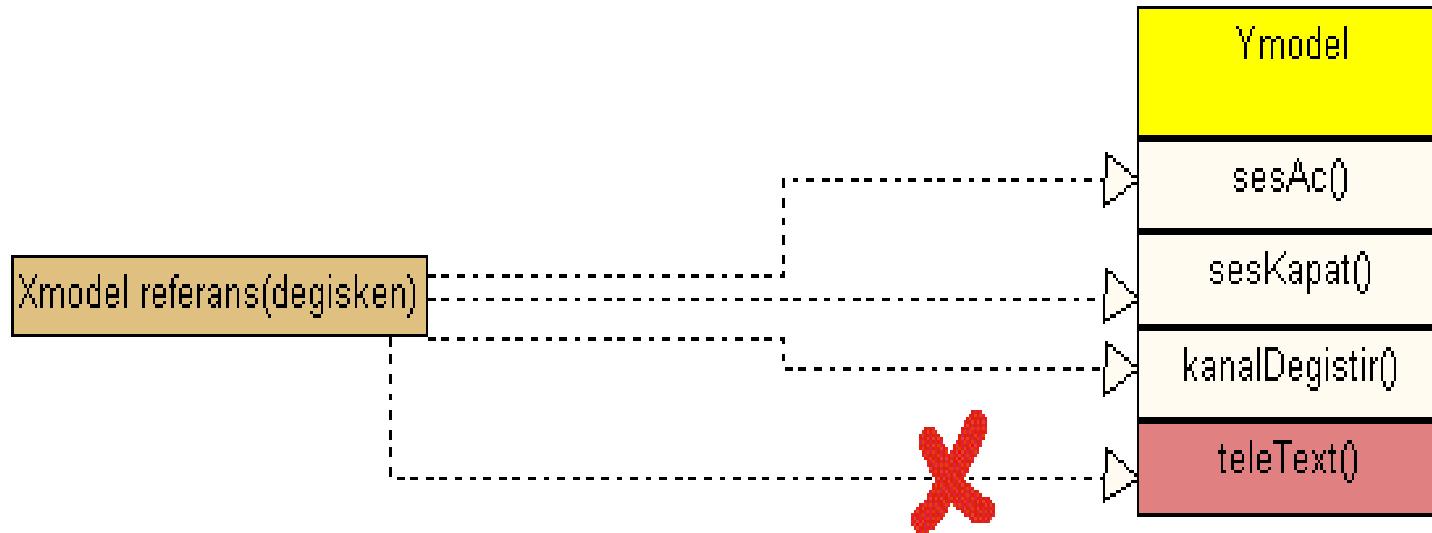
calis() çağrılmadan evvel
Futbolcu **calis()** 0 → **dikkat**
calis() çağrıldıktan sonra
Futbolcu yapılandırıcı
Futbolcu **calis()** 4



Örnek



Televizyon.java



Aşağıya Çevirim (*Downcasting*)

- 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.



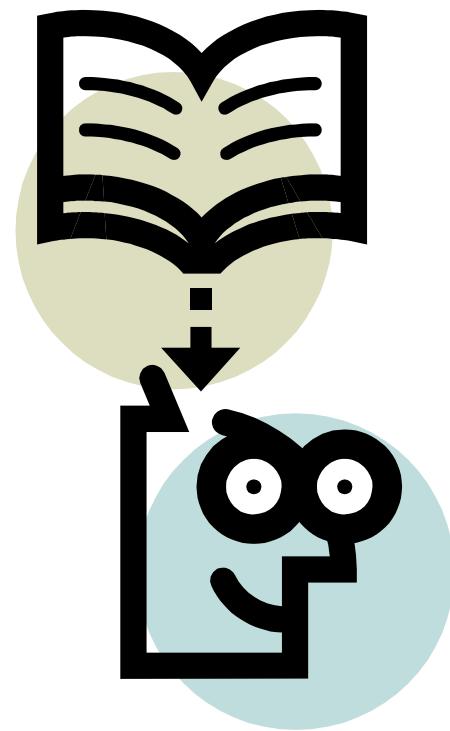
Televizyon2.java

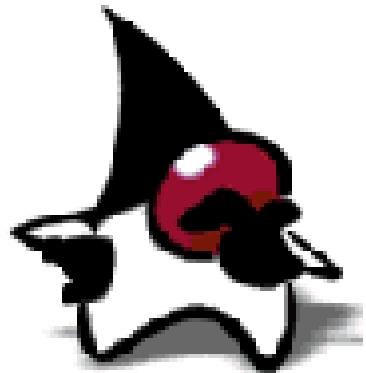
Sınıf tiplerinin çalışma anından tanımlanması (*RTTI : Run Time Type Identification*)



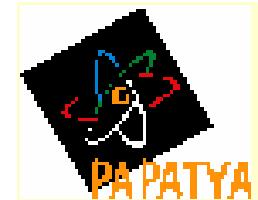
Televizyon3.java

Sorular ...





Arayüz (*Interface*)

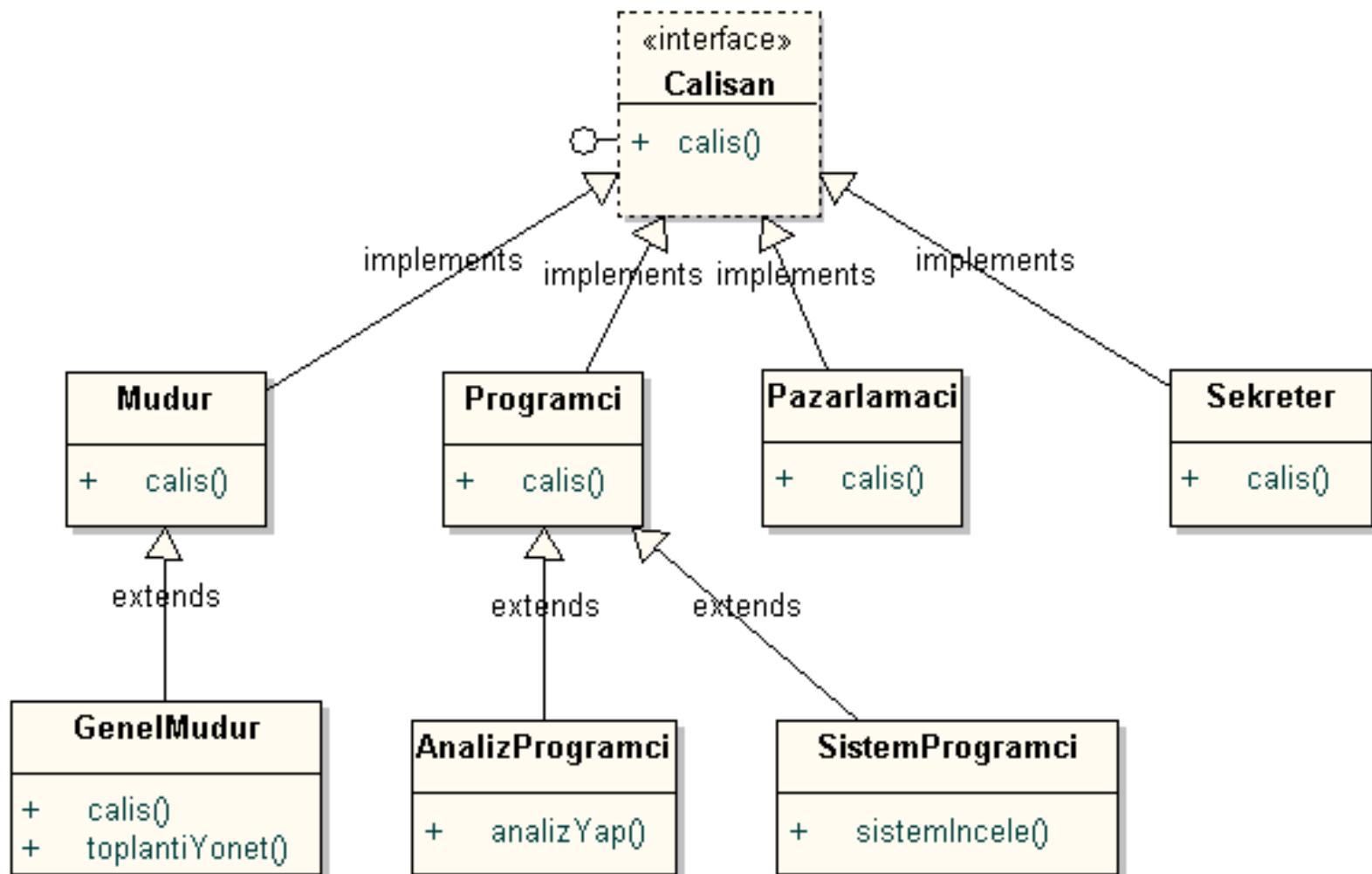


Arayüz (*Interface*)

- Arayüzler, soyut sınıfların bir üst modeli gibi düşünülebilir.
- Arayüzler tamamen birleştirici bir rol oynamaları için tasarlanmıştır.

Arayüz (Interface)

- Arayüzlerin içerisindeki gövdesiz (soyut) yordamlar (*methods*) otomatik olarak **public** erişim belirleyicisine sahip olurlar.



Örnek



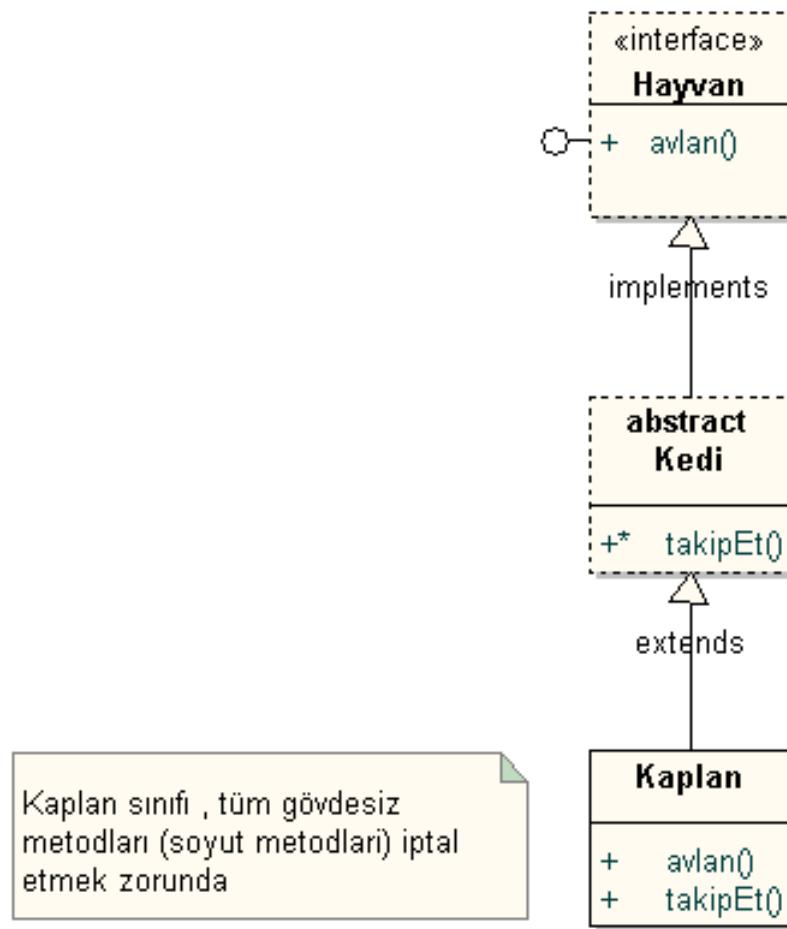
BuyukIsYeri.java

```
class Mudur implements Calisan {  
    public void calis() { // iptal etti (override)  
        System.out.println("Mudur Calisiyor");  
    }  
}
```

Arayüz(*Interface*) ve Soyut sınıflar(*Abstract classes*)

```
interface Hayvan {  
    public void avlan() ;  
}  
  
abstract class Kedi implements Hayvan {  
}
```

Bu örnek derlenir mi ?



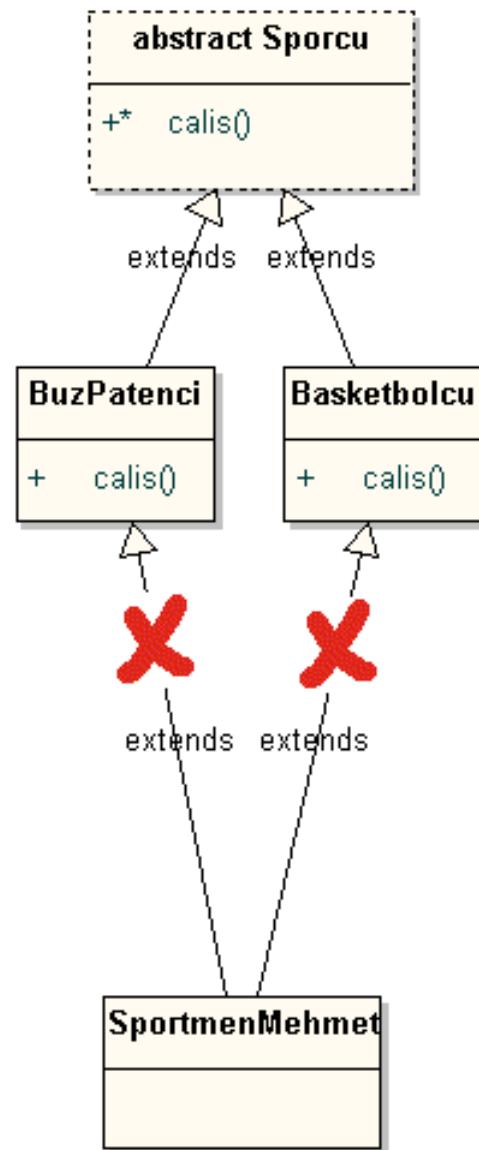
Örnek



Karism2.java

Arayüz(Interface) ile çoklu kalıtım(Multiple inheritance)

- Java Programlama dili çoklu kalıtımı (*multiple inheritance*) desteklemez.



Örnek - Hatalı

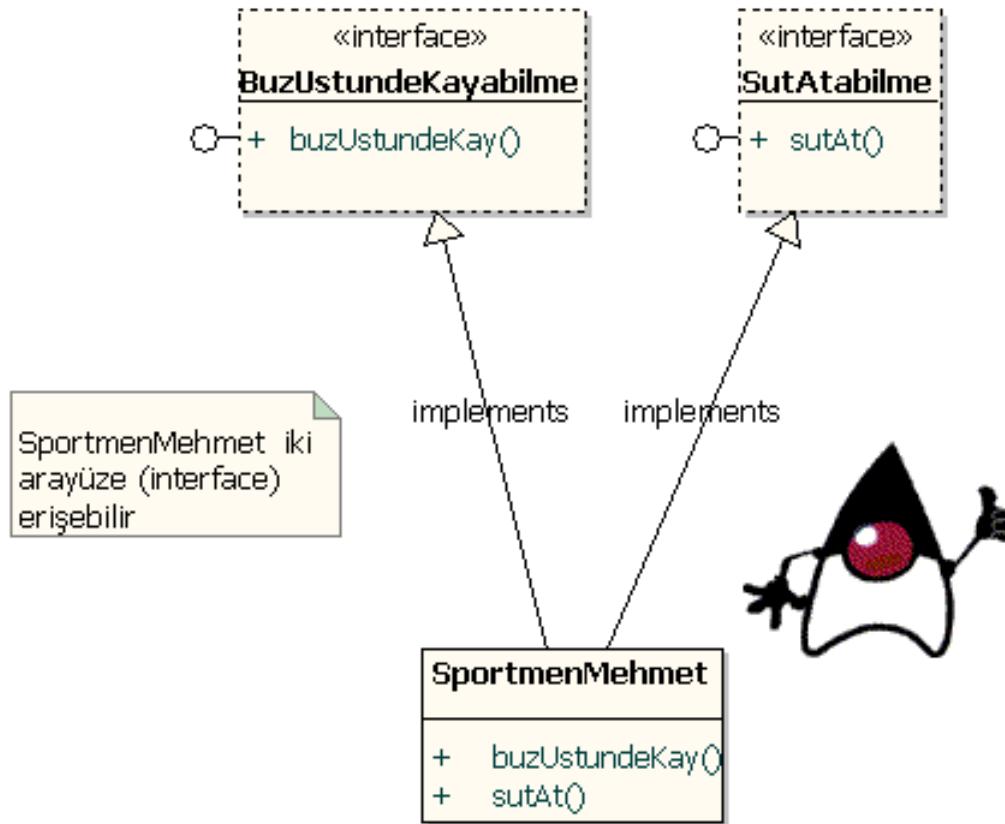


Spor.java

Sebep

```
Sporcu s = new SportmenMehmet(); // yukari cevirim  
s.calis(); // ??
```

Arayüz (*Interface*) ile çoklu kalıtımıma destek



Örnek

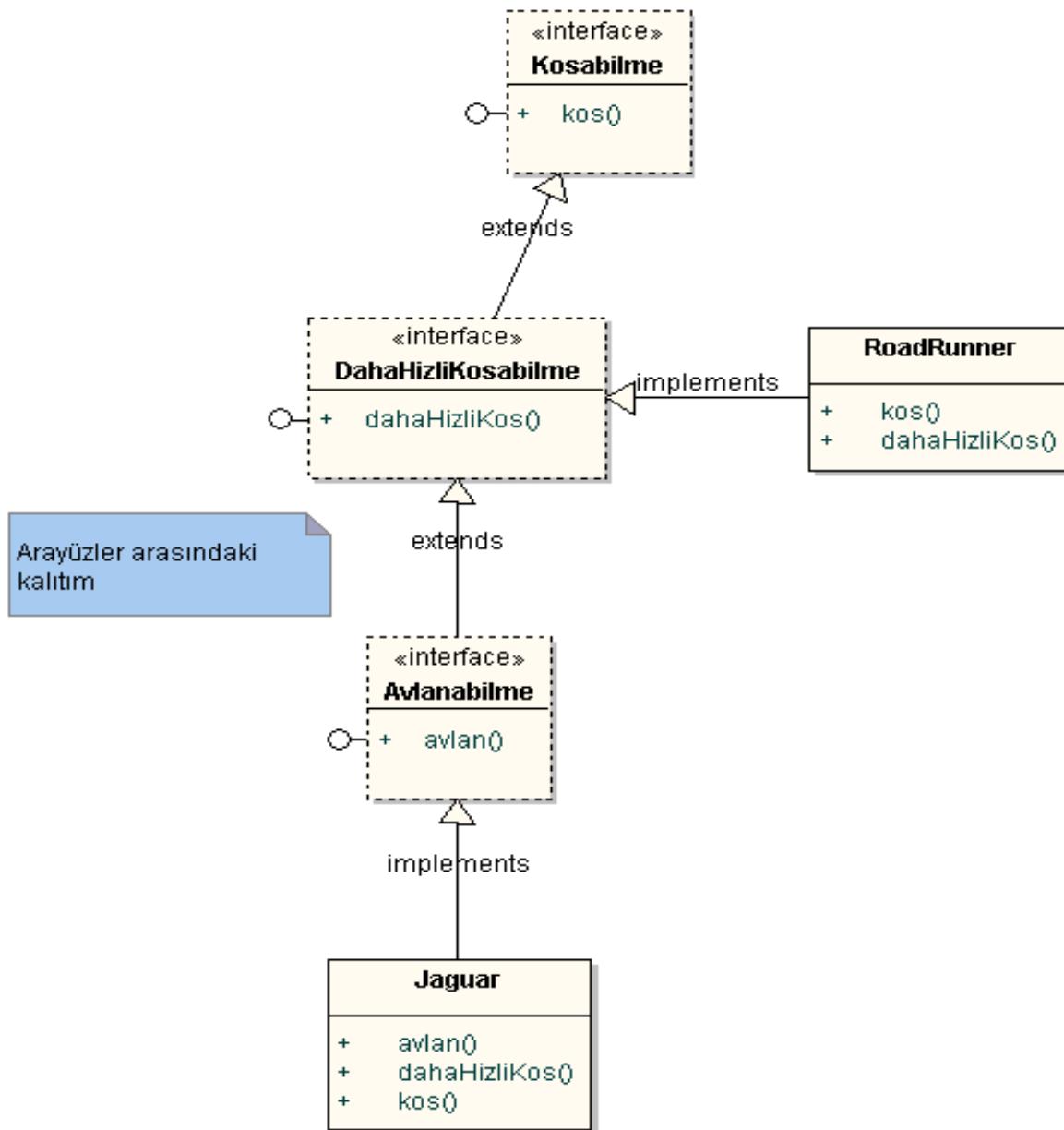
- *SportmenMehmet* belki hem *BuzPatenci* hem de *Basketbolcu* olamayabilir ama bunlara ait özelliklerini alabilir.



Spor2.java

Arayüzlerin kalıtım (*inheritance*) yoluyla genişletilmesi

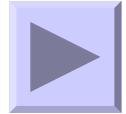
- Bir arayüz başka bir arayüzünden türetilabilir.
- Böylece arayüzler kalıtım yoluyla genişletilmiş olur.



Yakından bakılırsı

```
interface Avlanabilme extends DahaHizliKosabilme,Kosabilme {
    //..
}
```

Örnek



Jaguar.java

Çalışmalar

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



Cakisma.java (derlemeye çalışılırsa...)

```
public void hesapla(); // A1 arayüzüne ait  
public int hesapla(); // A3 arayüzüne ait
```

Arayüzün(*Interface*) içerisinde alan tanımlama

- Arayüzlerin içerisinde gövdesiz (soyut) yordamların dışında alanlar da bulunabilir.
- Bu alanlar, diğer uygulamalarda sabit olarak kullanılabilir.
- 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



AyBul.java

Arayüzün içerisinde tanımlanmış alanlara ilk değerlerinin verilmesi

- Arayüzlerin içerisinde tanımlanmış alanların ilk değerleri, çalışma anında (*runtime*) da verilebilir.



Test.java

Genel Bakış

- Arayüzler (*interface*) ve soyut (*abstract*) sınıfların bizlere sağlamak istediği fayda nedir ?

Genel Bakış

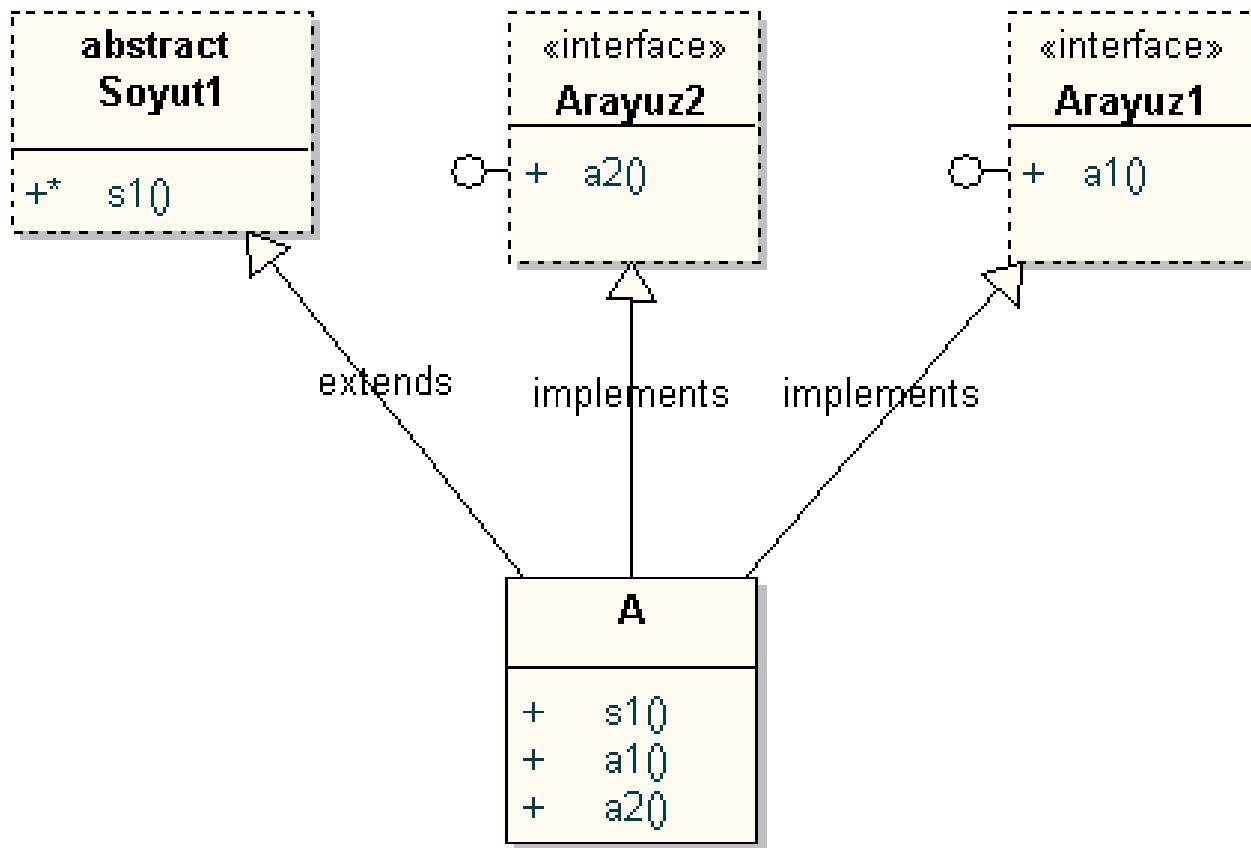
- Aslında ulaşılmak istenen amaç çoklu yukarı çevirimidir (upcasting).



GenelBakis.java

Yakından bakarsak

```
class A extends Soyut1 implements Arayuz1, Arayuz2 {  
    //.....  
}
```



```
Soyut1    soyut_1    =    new A();  
Arayuz1   arayuz_1   =   (Arayuz1) soyut_1; // tip degisimi  
Arayuz2   arayuz_2   =   (Arayuz2) soyut_1; // tip degisimi
```

Veya

```
Soyut1 soyut_1 = new A();  
Arayuz1 arayuz_1 = new A();  
Arayuz2 arayuz_2 = new A();
```

Dahili Sınıflar (*Inner Classes*)

- Dahili üye sınıflar
- Yerel sınıflar (*Local classes*)
- İsimlendirilmeyen sınıflar (*Anonymous classes*)

Dahili Üye Sınıflar

```
class CevreliyiciSinif {  
  
    class DahiliSinif {  
        //....  
    }  
  
    //...  
}
```

Örnek



Hesaplama.java

Dahili Üye Sınıflar ve Erişim

- Dahili üye sınıflara
 - public
 - friendly
 - protected
 - privateerişim belirleyicileri atanabilir.



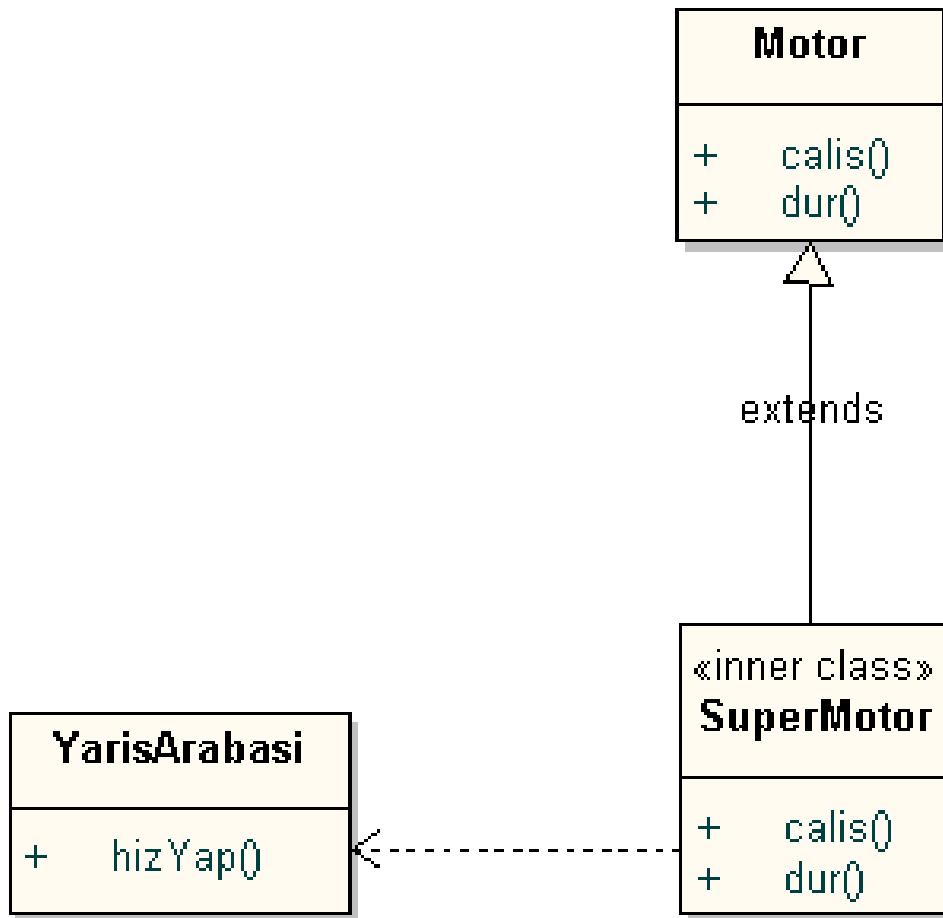
Hesaplama1.java



Hesaplama2Kullan.java

Türetilebilen Dahili Üye Sınıflar

- Dahili üye sınıflar, aynı normal sınıflar gibi başka sınıflardan türetilebilirler.
- Böylece çoklu kalıtım desteğinin bir benzerini (güvenli), Java programlama dilinde de bulabiliriz.



Örnek



YarisArabasi.java

Yerel Sınıflar (*Local Classes*)

- Yerel sınıflar
 - Yapılandırıcıların (*constructor*)
 - Sınıf yordamlarının (statik yordam)
 - Nesne yordamlarının
 - Statik alanlara toplu değer vermek için kullandığımız statik bloğun
 - Statik olmayan alanlara toplu değer vermek için kullandığımız bloğun içerisinde tanımlanabilir.

Yerel Sınıflar

```
public class Sinif {  
    public void yordam() {  
        public class YerelSinif {  
            //...  
        }  
    }  
}
```

Yerel sınıflara ait ilk özellikler

- Yerel sınıflar tanımlandıkları yordamın veya blogun dışından erişilemezler.
- Yerel sınıflar başka sınıflardan türetilabilir veya arayzlere (*interface*) erişebilir.
- Yerel sınıfların yapılandırıcıları olabilir.



Hesaplama6.java

Yerel Sınıflara Ait İlk Özellikler - Devam

- Yerel sınıflar, içinde bulundukları yordamın sadece **final** olan değişkenlerine ulaşabilirler.
- Yerel sınıflar, **statik** veya **statik olmayan** yordamların icerisinde tanımlanabilirler.
- Yerel sınıflar, **private**, **protected** ve **public** erişim belirleyicisine sahip olamazlar sadece **friendly** erişim belirleyicisine sahip olabilirler.
- Yerel sınıflar, **statik** olarak tanımlanamaz.



Hesaplama7.java

İsimsiz Sınıflar (*Anonymous Classes*)

- Diğer dahili sınıf çeşitlerinde olduğu gibi, isimsiz sınıflar direkt **extends** ve **implements** anahtar kelimelerini kullanarak diğer sınıflardan türetilemez ve arayzlere erişemez.
- Isimsiz sınıfların herhangi bir ismi olmadığı için yapılandırıcısı da (*constructor*) olamaz.



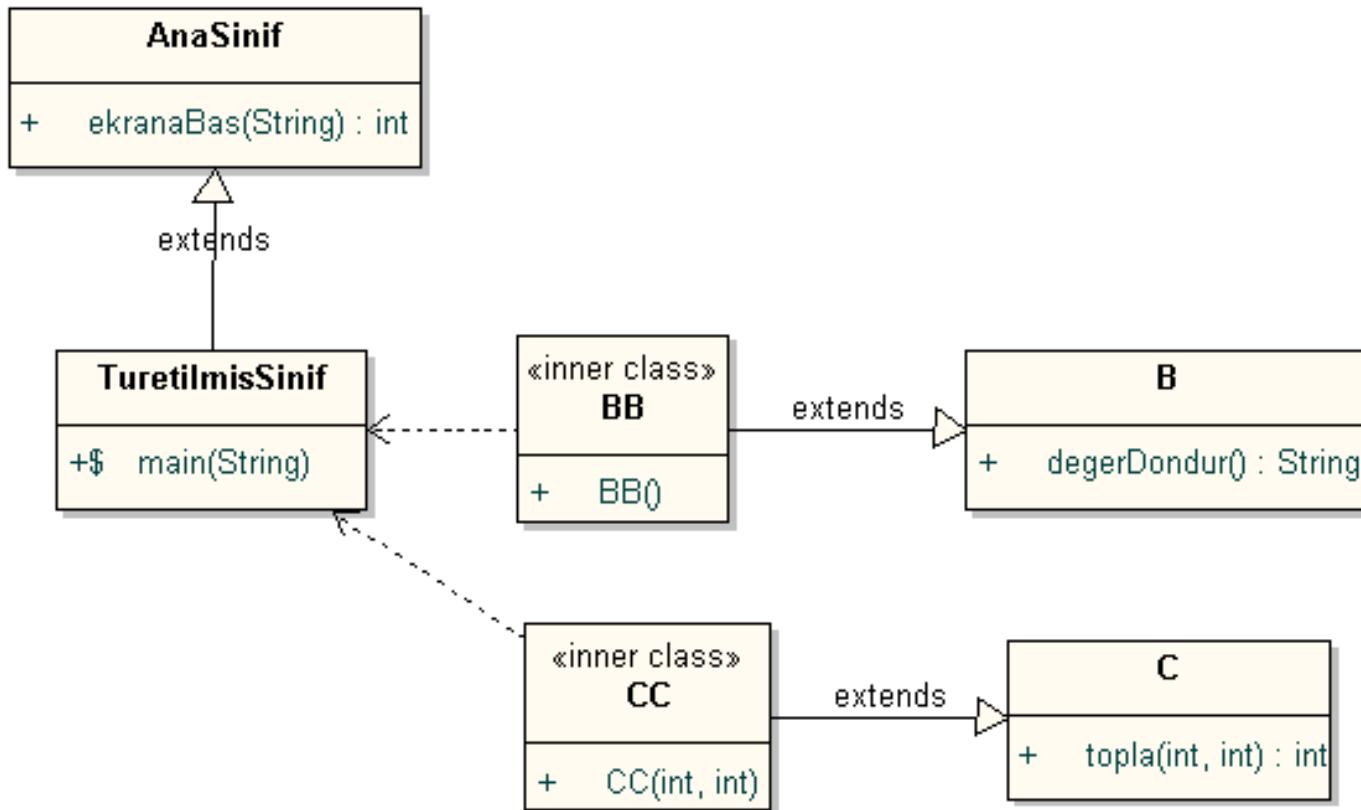
Hesaplama8.java

Yakından bir bakış

```
return new Toplayici() {  
    public int hesaplaMaYap() {  
        // final olan yerel degiskenlere ulasabilir.  
        return a + b ;      }  
} ; // noktali virgül şart
```

Neden Dahili sınıflar?

- Arayüzler ile çoklu kalıtım (*multiple inheritance*) desteğini kısmen bulabiliyorduk ama bu tam değildi.
- Dahili sınıfların var olmasındaki neden çoklu kalıtımı tam desteği sağlamaktır.

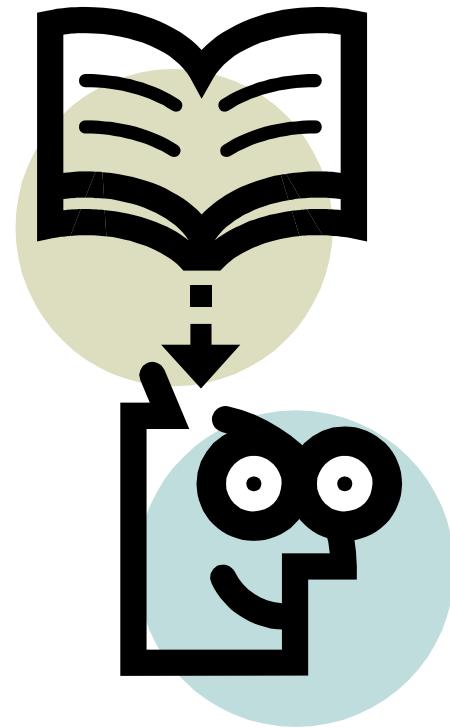


Örnek



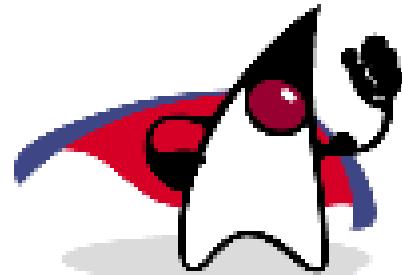
TuretilmisSinif.java

Sorular ...





İstisnalar (Exceptions)



İstisna deyince aklınıza ne geliyor ?

- “Diğerlerinin yazdığını programda hata olabilir ama benim yazdığım programda hata olmaz... ” - Anonim

Tam bir uygulama yazmak nasıl olur ?

- Bir uygulamanın, üzerine düşen işleri yapması onu tam bir uygulama yapar mı ?

Tam bir uygulama

- Doğruluk
- Sağlamlık

Doğruluk- ilk şart

- İki sayıyı bölmeye yarayan bir uygulama geliştirmeniz istendi.
- $A / B = \text{Sonuç} \rightarrow$ çok kolay

Sağlamlık-unutulan şart

- Dışarıdan istenmeyen veriler girildiği zaman uygulamanız nasıl bir davranış sergiliyecektir ?
 - $5 / 0 = ?$
 - $10 / \text{elma} = ?$
 - $\text{armut} / \text{erik} = ?$
- Uygulamanız aniden kapanacak mı ? Yoksa bunları bölmeye mi kalkacak ?

Java ve sağlamlık şarttı

- Java Programlama dili **oluşabilecek** olan istisnalara karşı önlem alınmasını ister.
 - Açılmak istenen dosya yerinde olmayabilir.
 - Ağ (*network*) bağlantısı kopmuş olabilir.
 - Okunmak istenen dosyanın içi boş olabilir.

İstisna nasıl oluşabilir ?



DiziErisim.java

İstisna yakalama mekanizması

```
try {  
    // istisnaya sebebiyet verebilecek olan kod  
} catch (Exception1 e1) {  
    //Eğer Exception1 tipinde istisna fırlatılırsa buraya  
} catch (Exception2 e2) {  
    //Eğer Exception2 tipinde istisna fırlatılırsa buraya  
}
```

Örnekler



DiziErisim2.java



DiziErisim3.java

İstisna İfadeleri

- Bir yordamın iki seçeneği vardır
 - Oluşan istisnayı yakalayabilir (hata yakalama mekanizması kullanarak)
 - Oluşan istisnayı bir üst kısma fırlatabilir.

Örnek-1

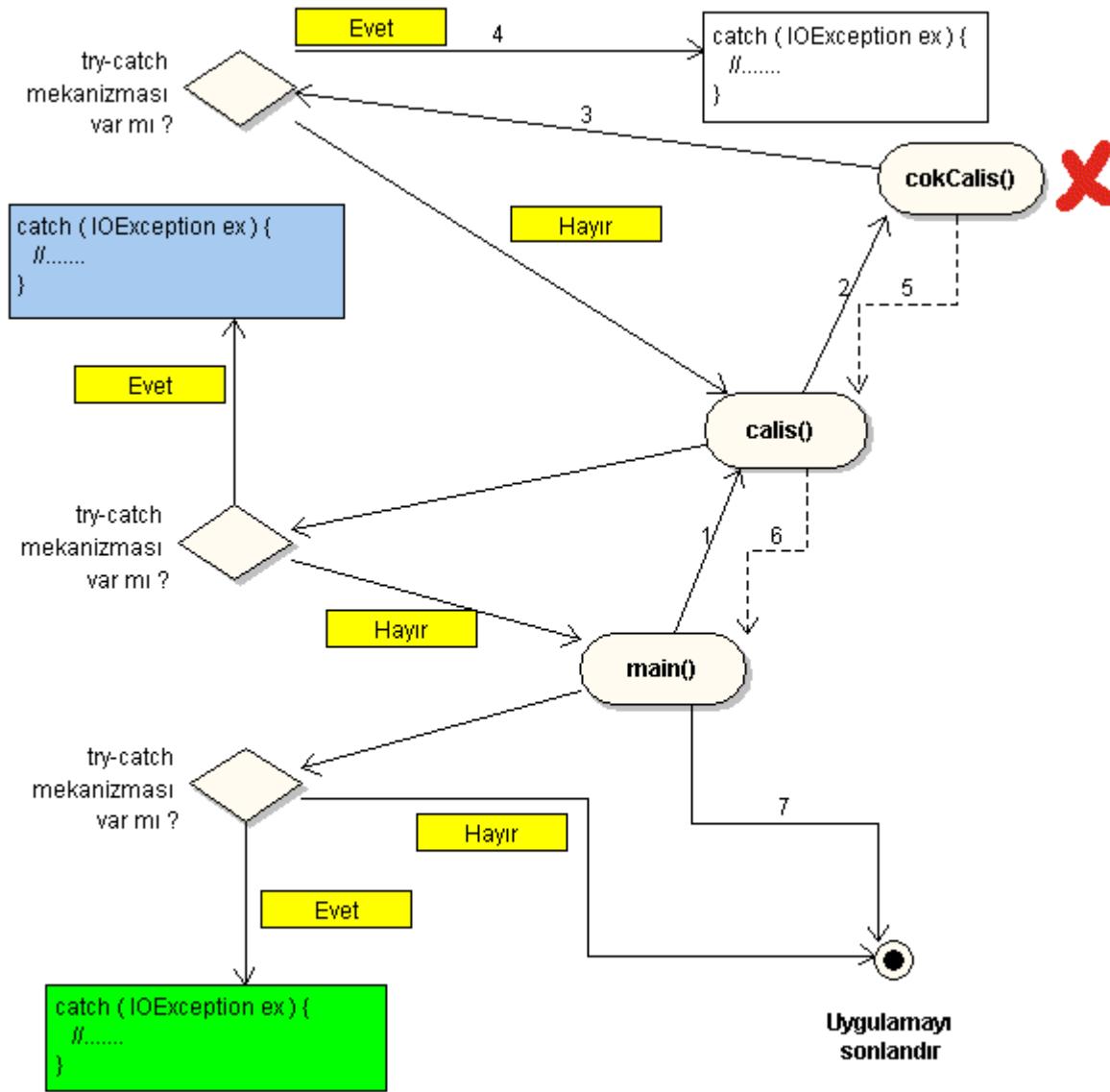


IstisnaOrnek1.java

Örnek-2



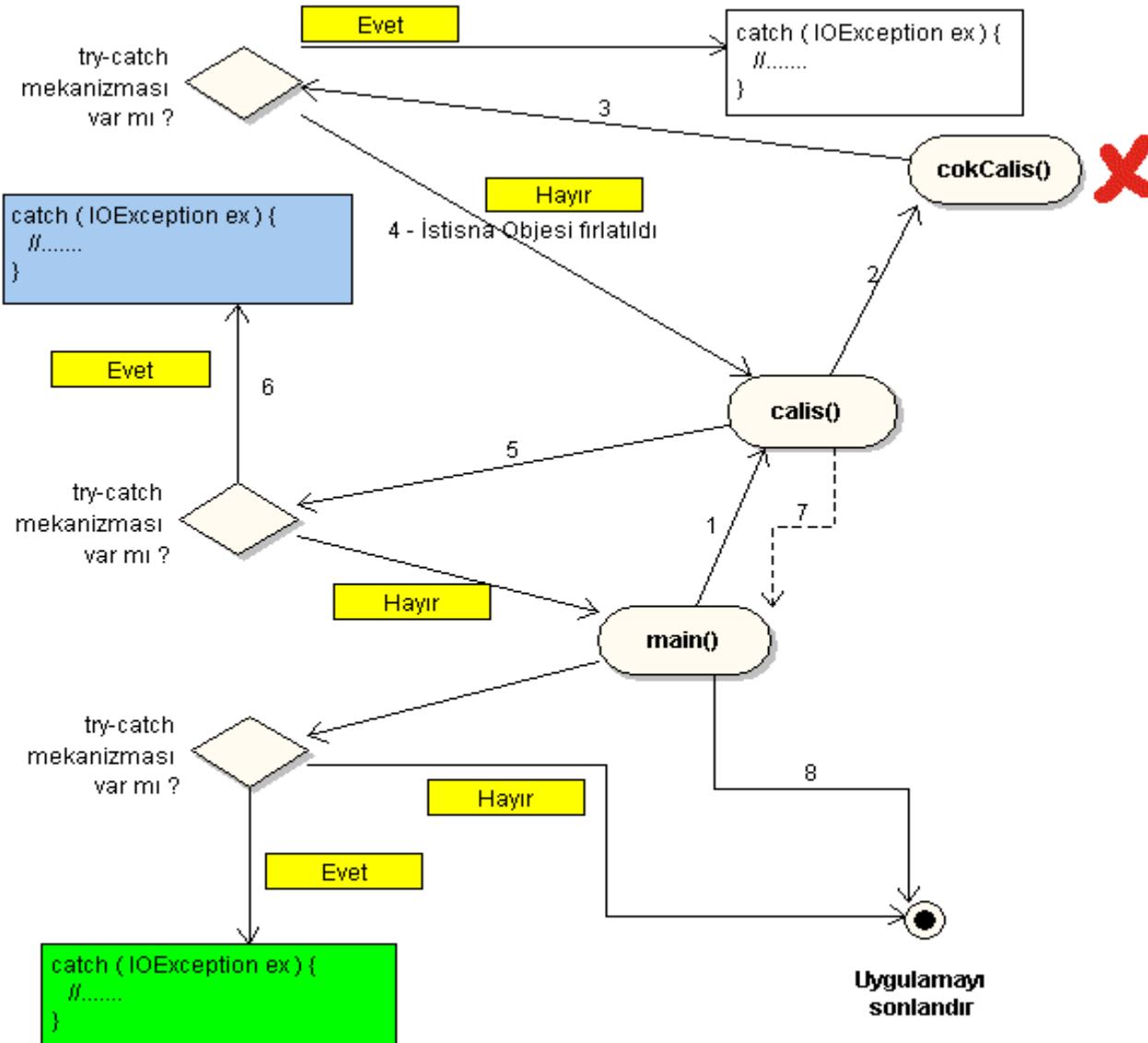
IstisnaOrnek2.java



Örnek-3



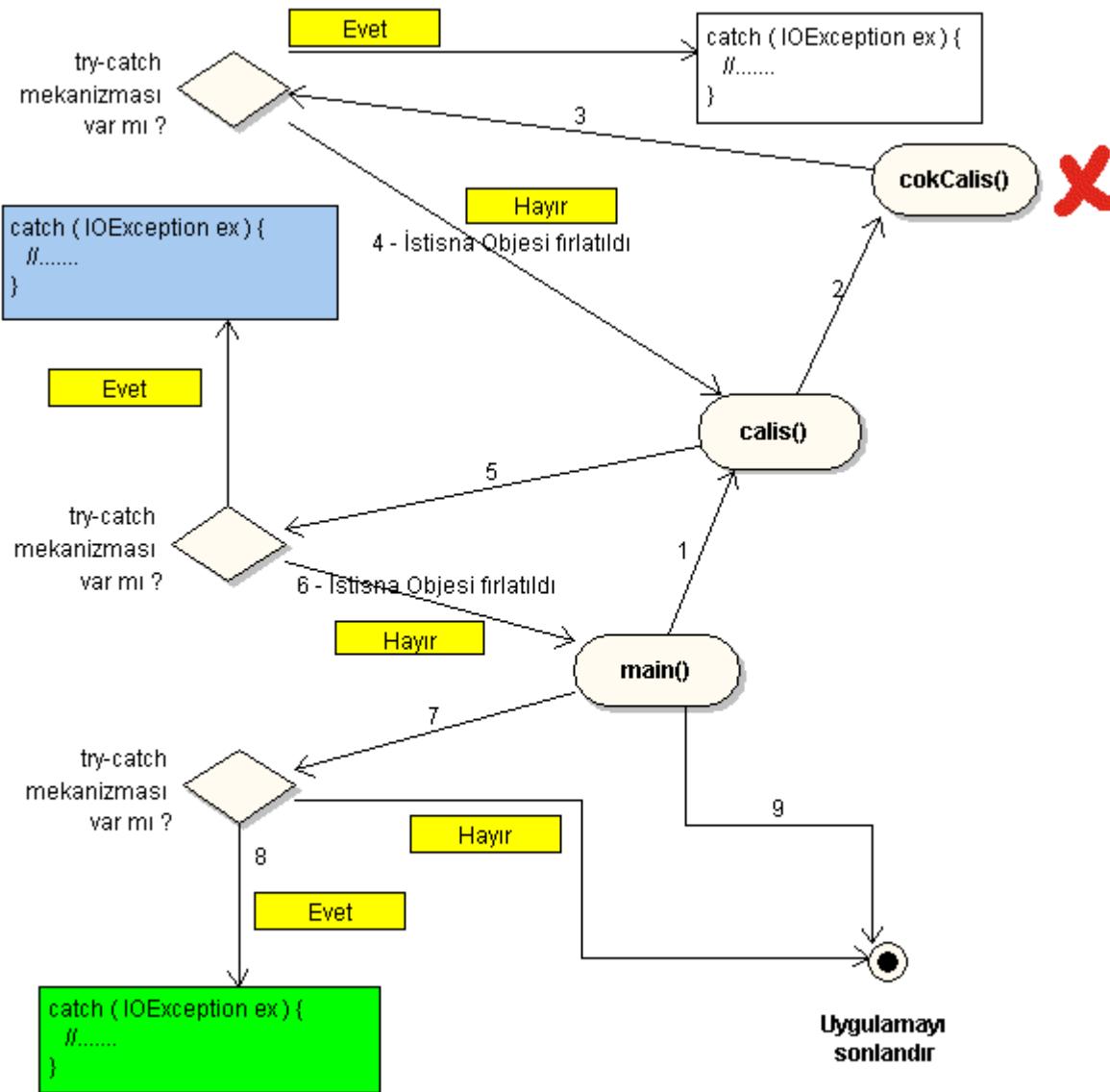
IstisnaOrnek3.java



Örnek-4



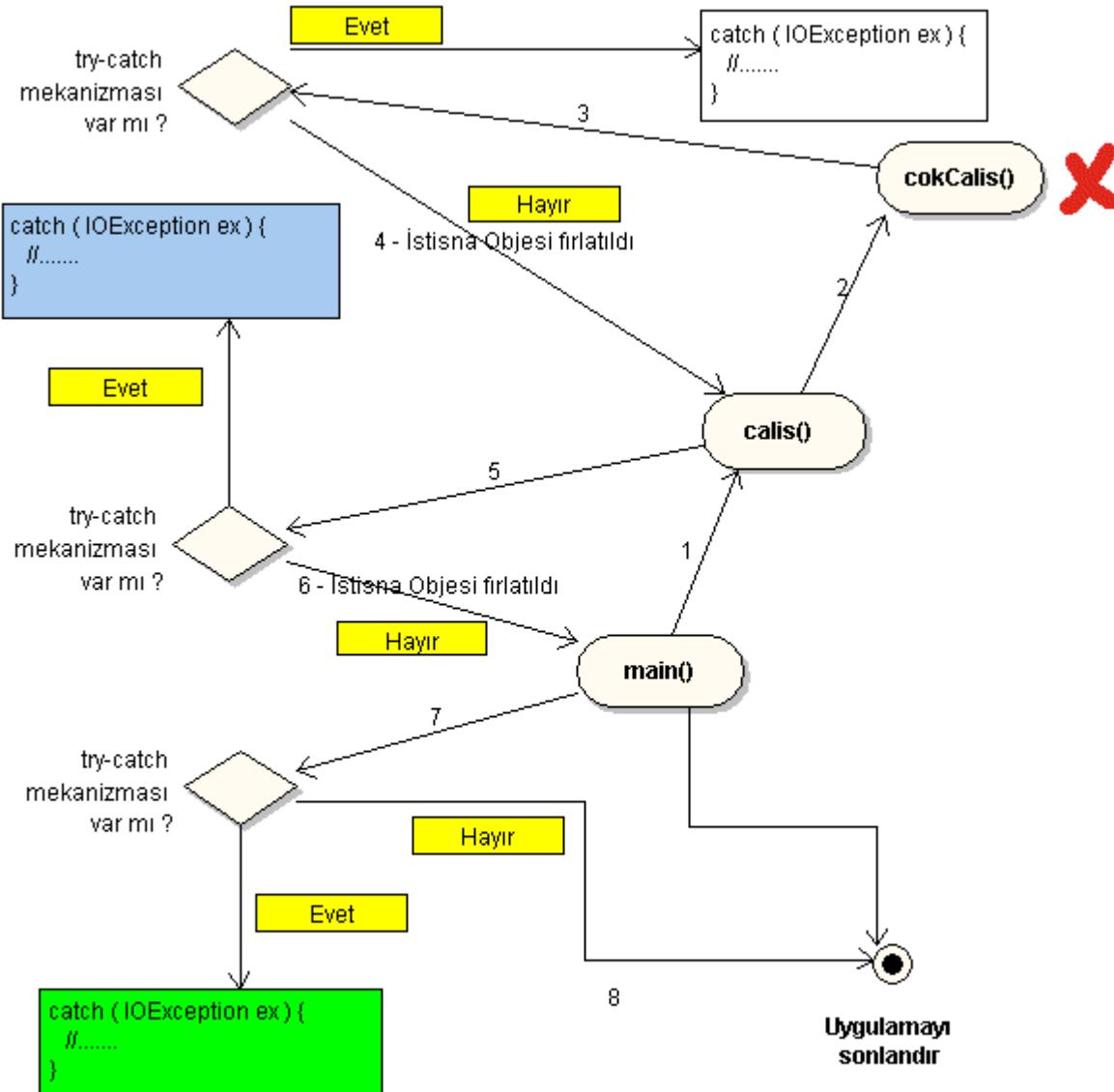
IstisnaOrnek4.java



Örnek-5

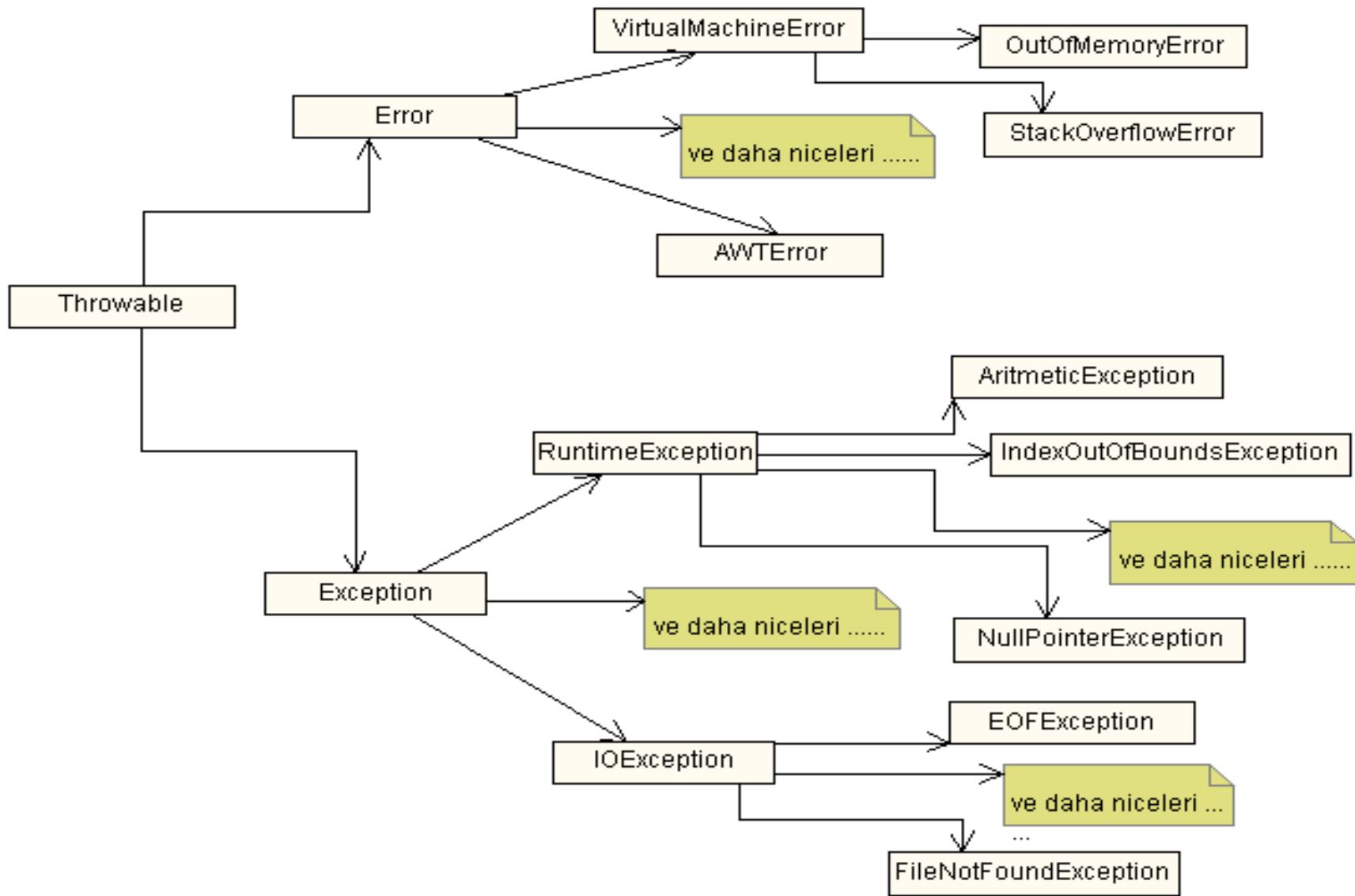


IstisnaOrnek5.java



İstisna tip hiyerarşisi

- Nasıl olur da *java.io.IOException* istisna tipi, *java.io.FileNotFoundException* istisna tipini kapsayabilir ?



- Error
- RuntimeException
- Ve diğer Exception tipleri (önceden tedbir alınmaları gereklidir)

- *Error* istisna tipi ölümcül bir hatayı işaretettir ve telafisi çok zordur; neredeyse imkansızdır.
- Örneğin *OutOfMemoryError* istisna tipi, bellekten dolayı bir istisna meydana gelmiş ise uygulamanın buna müdahale edip düzeltmesi imkansızdır.

RuntimeException

- *RuntimeException* istisna tipleri, eğer uygulama normal seyrinde giderse ortaya çıkmaması gereken istisna tipleridir.
- *ArrayIndexOutOfBoundsException* istisna tipi, bir dizinin olmayan elemanına eriştiğimiz zaman ortaya çıkan bir istisnadır.
- *RuntimeException* istisna tipleri, yanlış kodlamadan dolayı ortaya çıkan bir istisnadır.

Ve diğer Exception tipleri

- Bu istisna tipleri çevresel koşullardan dolayı meydana gelebilir.
- Örneğin erişilmeye çalışılan dosyanın yerinde olmaması (*FileNotFoundException*) veya ağ (*network*) bağlantısının kopması sonucu ortaya çıkabilecek olan istisnalarıdır. Bu istisnalar için önceden tedbir alınması şarttır.

Tüm diğer Exception istisna tiplerini yakalamak

```
catch (Exception ex) {  
    // .....  
}
```

RuntimeException istisna tipleri

- *DiziErisim.java* uygulama örneğimiz içerisinde istisna oluşturma riski olmasına rağmen nasıl oldu da Java buna kızmayarak derledi ? Peki ama *IstisnaOrnek1.java* uygulamasını niye derlemedi ?

RuntimeException istisna tipleri

- Java Programlama Dili, *RuntimeException* istisna tiplerini yakalamak için bir baskı yapmaz.

RuntimeException istisna tipleri nelerdir ?

- *ArithmeticException*
- *NullPointerException*
- *NegativeArraySizeException*
- *ArrayIndexOutOfBoundsException*
- *SecurityException*

AritmeticException

- Bir sayının sıfıra bölünmesiyle ortaya çıkabilecek olan *RuntimeException* istisna tipi.

```
int i = 16 / 0 ; // AritmeticException ! hata !
```

NullPointerException

- Bir sınıf tipindeki referansı, o sınıfa ait bir nesneye bağlamadan kullanmaya kalkınca alınabilecek bir istisna tipi.

```
String ad ;  
System.out.println("Ad = " + ad.trim() ) ; // ! hata !
```

Bu hatayı almamak için ;

```
String ad = " Java Kitap Projesi " ; // bağlama işlemi  
System.out.println("Ad = " + ad.trim() ) ; //dogru
```

NegativeArraySizeException

- Bir diziyi negatif bir sayı vererek oluşturmaya çalışırsak bu istisna tipi ile karşılaşırız.

```
int dizi[] = new dizi[ -100 ] ; //! hata !
```

ArrayIndexOutOfBoundsException

- Bir dizinin olmayan elemanına ulaşmak istediği zaman karşılaşılan istisna tipi
- Daha detaylı bilgi için ***DiziErisim.java*** uygulama örneğini incelenebilir.

SecurityException

- Genellikle tarayıcı (*browser*) tarafından fırlatılan bir istisna tipidir. Bu istisnaya neden olabilecek olan sebepler aşağıdaki gibidir ;
 - Applet içerisinde, yerel bir dosyaya erişmek istediği zaman.
 - Applet'in indirildiği sunucuya değil de değişik bir sunucuya ağ (*network*) bağlantısı kurulmaya çalışıldığı zaman.
 - Applet'in kendi içerisinde başka bir uygulama başlatmaya çalıştığı zaman.

- Bir istisna nesnesinden bir çok veri elde edilebilir.
- Örneğin istisna oluşumunun yol haritası izlenebilir veya istisna oluşana kadar hangi yordamlar çağrılmış gibi değerli bilgiler görülebilir.

İstisna Mesajları

- **String getMessage()** : Oluşan istisnaya ait bilgileri **String** tipinde geri döner.
- **String getLocalizedMessage()** : Bu yordam, *Exception* sınıfından türetilmiş alt sınıflar tarafından iptal edilebilir (*override*).
- **String toString()** : Oluşan istisna hakkında kısa bir açıklamayı **String** tipinde geri döner.
 - Oluşan istisna nesnesinin tipini ekrana basar
 - ":" iki nokta üst üste koyar ve bir boşluk bırakır.
 - Son olarak **getMassege()** yordamı çağrıılır ve buradan - eğer bilgi varsa ekrana basılır.

Örnek-1



IstisnaMetodlari.java

- **Throwable getCause()** : Bu yordamın işe yaraması için istisna sınıfına ait yapılandırıcının içerisinde bu istisnaya sebebiyet vermiş olan istisna tipini yerleştirmek gerekmektedir.



IstisnaMetodlari2.java

- **Throwable initCause(Throwable cause) :**
İki yarı istisna tipini birleştirmeye yarar. Eğer bir istisna Throwable(Throwable) veya Throwable(String, Throwable) ile oluşturulmuş ise **initCause()** yordamı çağırılamaz.



IstisnaMetodlari3.java

İstisna Mesajları

- **printStackTrace()** : *Throwable* sınıfının bu yordamı sayesinde oluşan bir istisnanın yol haritasını görebiliriz.
- **printStackTrace(PrintStream s)** : *PrintStream* sınıfına ait nesne kullanılarak oluşan istisnanın yol haritasını konsol yerine başka bir yere bastırmanız mümkündür.
- **printStackTrace(PrintWriter s)** : *PrintWriter* sınıfına ait nesne kullanılarak, oluşan istisnanın yol haritasını konsol yerine başka bir yere bastırmanız mümkündür. JSP ve Servlet lerde kullanılabilir.



IstisnaMetodlari4.java

Kendi İstisnalarımızı Nasıl Oluşturabiliriz?

- Java'nın kendi içerisinde tanımlanmış istisna tiplerinin dışında bizler de kendimize özgü istisna tiplerini oluşturup kullanabiliriz.
- Sonuçta istisnalar da birer nesnedir ve kendilerine has durumları olabilir.



BenimHatam.java



SeninHatan.java

Örnek



Kobay.java

Ekran Çıktısı

```
-----  
Hata Olustu-1:sonuc eksi  
! onemli hata !
```

2

```
-----  
Hata Olustu-2:SeninHatan:  
b parametresi sifir geldi
```

```
-----  
Hata Olustu-2:SeninHatan
```

- Bir işlemin her koşulda (istisna olsun ya da olmasın) kesin olarak yapılmasını istiyorsak **finally** blogu kullanmalıyız.



FinallyOrnek1.java



FinallyOrnek2.java

- **finally** blogu her zaman çalıştırılır.
- Örneğin bir yordam hiçbir şey döndürmüyorsa (*void*) ama bu yordamın içerisinde return ifadesi kullanılmış ise, **finally** blogu, bu **return** ifadesi devreye girmeden hemen önce çalıştırılır.



ReturnOrnek.java

Dikkat System.exit();

- Eğer *System* sınıfının statik bir yordamı olan **exit()** çağrılsa **finally** bloğuna hiç girilmez.



SystemExitOrnek.java

İstisnanın Tekrardan Fırlatılması

- Oluşan bir istisnayı **catch** bloğunda yakaladıktan sonra tekrardan bir üst kısma fırlatmanız mümkündür.

Gösterim

```
try {  
    // riskli kod  
} catch (Exception ex) {  
    System.out.println("istisna yakalandi: " + ex);  
    throw ex; // dikkat  
}
```

Örnek



TekrarFirlatimOrnek1.java

- Oluşan bir istisna her zaman fırlatılamayabilir.



FirlatimOrnek1.java



FirlatimOrnek2.java

Çözüm



FirlatımOrnek3.java

İptal Etme (*Override*) ve İstisnalar

- İptal etme şartları
 - İptal eden yordamın, iptal edilen yordam ile aynı parametrelere,
 - Aynı isme,
 - Üst sınıfı ait yordamın erişim belirleyicisinden daha erişilebilir veya aynı erişim belirleyicisine sahip olması gereklidir.

Örnek



AB.java



CD.java



EF.java

İstisnaların Sıralanması

- Bir istisna **catch** bloğunda veya **catch** bloklarında yakalanırken, istisnaların hiyererşik yapılarına dikkat edilmelidir.



IstisnaSiralamasi.java

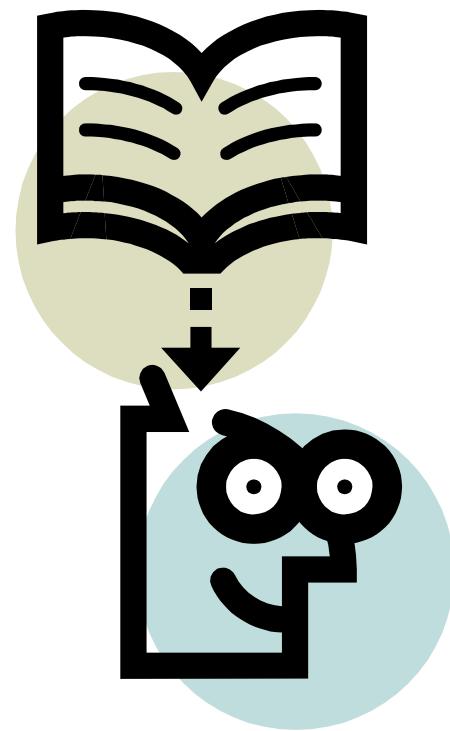


IstisnaSiralamasi2.java

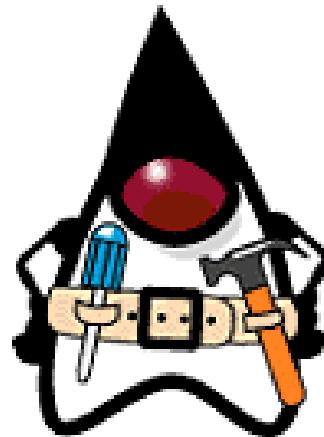


IstisnaSiralamasi3.java

Sorular ...



JAVA'DA GİRİŞ/ÇIKIŞ İŞLEMLERİ

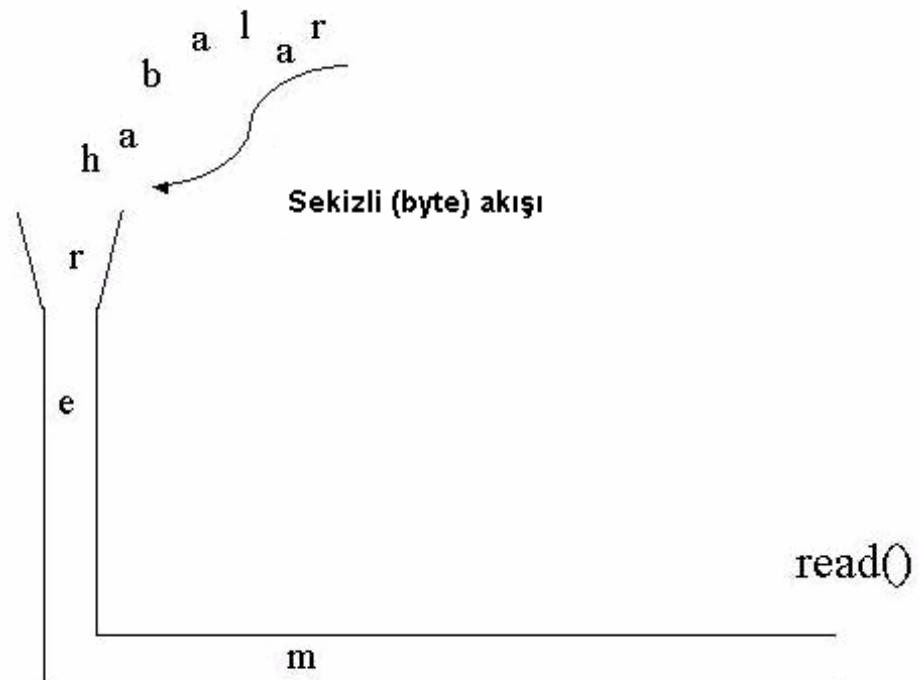


G/Ç işlemleriyle uygulamalara neler yaptırabiliriz...

- Dosya işlemleri
 - Dosyadan okuma
 - Dosyaya yazma
- Verilerin Aktarılması
 - Ağ (*network*) üzerinden
- Nesnelerin kalıcı ortama yazılması
(*Serialization*)
- Rasgele erişimli dosyalar (*Random access file*)

Irmak (*Stream*)

- Uygulama ile kaynak arasındaki yol/bağlantı
 - Sekizli (*byte*) ırmakları
 - Karakter ırmakları



Gelen sekizli(*byte*) ırmakları

- **Gelen:** Kaynaktan → uygulamaya doğru bir akışı belirtir.
- **Sekizli (*byte*):** Kurulan bağlantı içerisinde sekizli(*byte*) tipinde verilerin akacağını belirtir.
- **Irmak:** Kaynak ile uygulama arasında kurulan bağlantı.

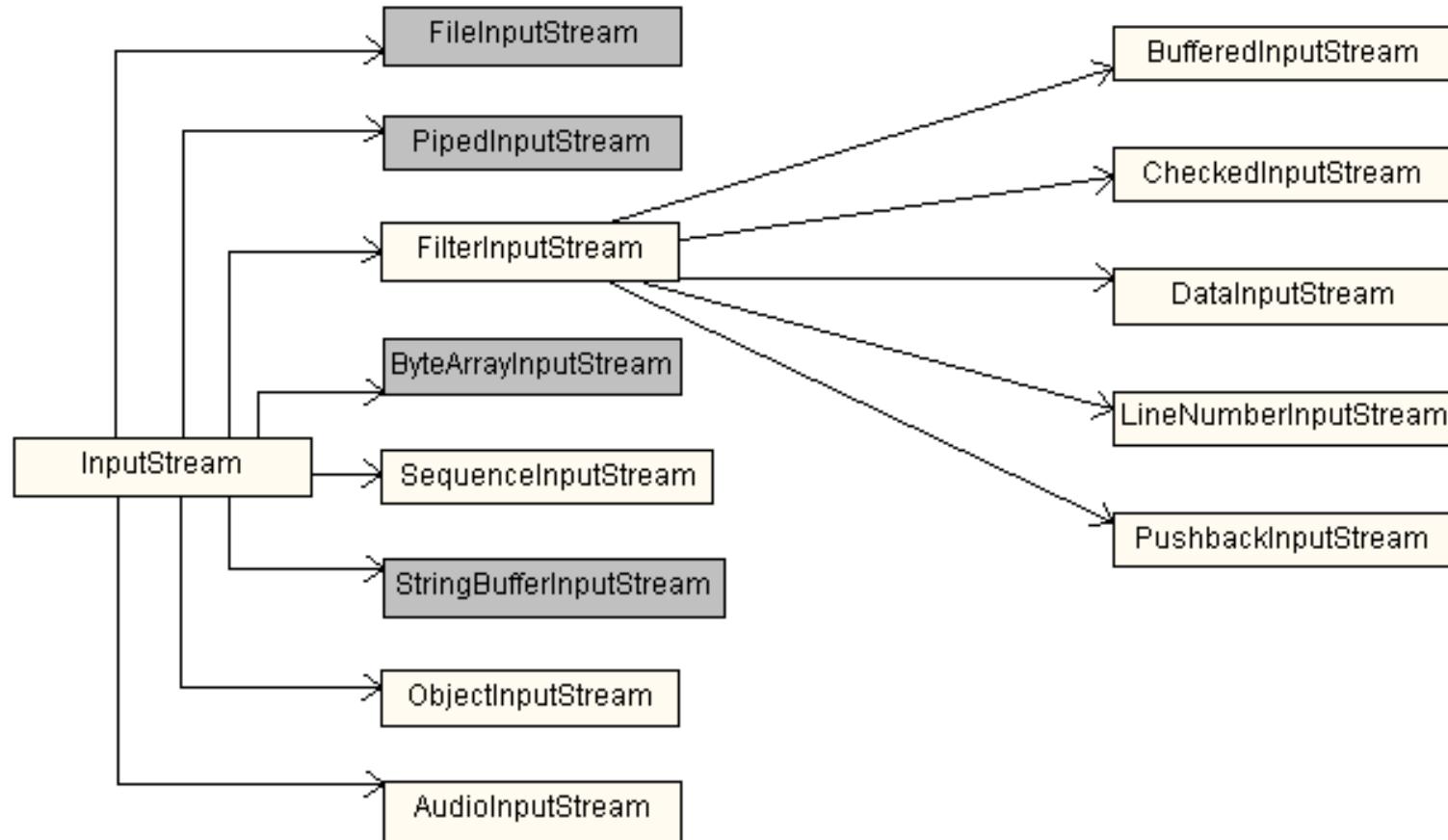
Sekizli İrmakları (*Byte Streams*)

- Sekizli(*byte*) ırmakları üzerinde iş yapabilmek için tasarlanmış sınıflar Java 1.0'dan itibaren mevcuttur.
 - *java.io.InputStream* (Gelen sekizli ırmakları)
 - *java.io.OutputStream* (Giden sekizli ırmakları)

Önemli bir nokta

- *java.io* paketinin altındaki bir sınıfın ismi eğer *InputStream* veya *OutputStream* ile bitiyorsa o zaman;
 - “Bu sınıf, sekizli ırmakları (*byte streams*) üzerinde işlem yapması için tasarlanmıştır” sonucu çıkartılabilir.
 - *FileInputStream sınıfı*
 - *BufferedOutputStream sınıfı*
 - gibi...

InputStream soyut sınıfı ve bu soyut sınıfından türemiş alt sınıflar



InputStream soyut sınıfına ait yordamlar (methods) - I

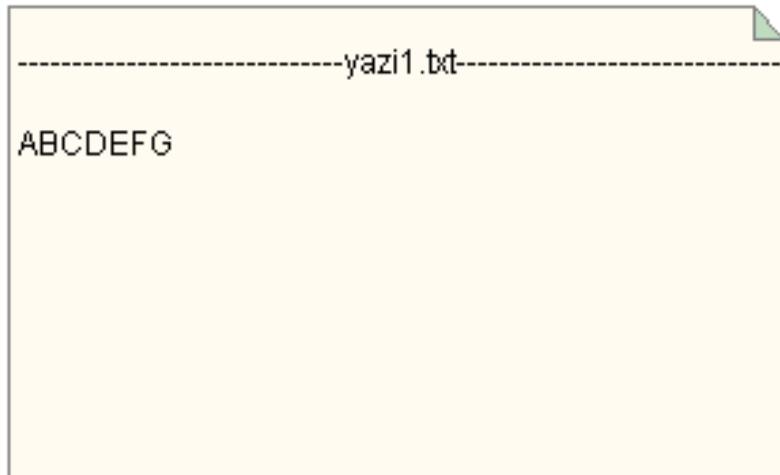
available()	Gelen sekizli (bayt) ırmağı içerisindeki <u>okunabilecek</u> olan verilerin sayısını belirtir. Bu yordam alt sınıflar tarafından iptal edilmelidir.
close()	Kaynak ile uygulama arasında kurulmuş olan bağlantıyı kapatır.
mark(int okumalimiti)	İçsel dizi kullanılarak tamponlanmış olan veriler içerisinde istenilen pozisyonu işaret koyar.
markSupported()	Gelen sekizli ırmağı için mark() ve reset() yordamlarının çalışabilirliğini test eden yordamdır. Eğer bu yordamlar, okunan sekizli(<i>byte</i>) ırmağı üzerinde çalışabilir ise true , değilse false döner.

InputStream soyut sınıfına ait yordamlar (methods) – II

read()	Gelen sekizli ırmağı içerisinde tek bir veriyi okuyan yordam. Bu yordamın türemiş alt sınıflar tarafından iptal edilmesi (<i>override</i>) gereklidir.
read(byte[])	Gelen sekizli ırmağı içerisinde, parametre olarak gönderilen dizi uzunluğu kadar veriyi okur ve bunu ilgili dizinin sıfırıncı indeksinden itibaren yine o dizİYE yerleştirir.
read(byte[] b, int off, int len)	Gelen sekizli ırmağı içerisinde, parametre olarak gönderilen dizi uzunluğu kadar veriyi okur ve bunu ilgili dizİYE verilen ölçülerde (off, len) yerleştirir.
reset()	mark() ile İşaretlenen pozisyonu geri döndüren yordam.
skip(int n)	Gelen sekizli ırmağı içerisinde <u>n</u> uzunlığında atlama yapar.

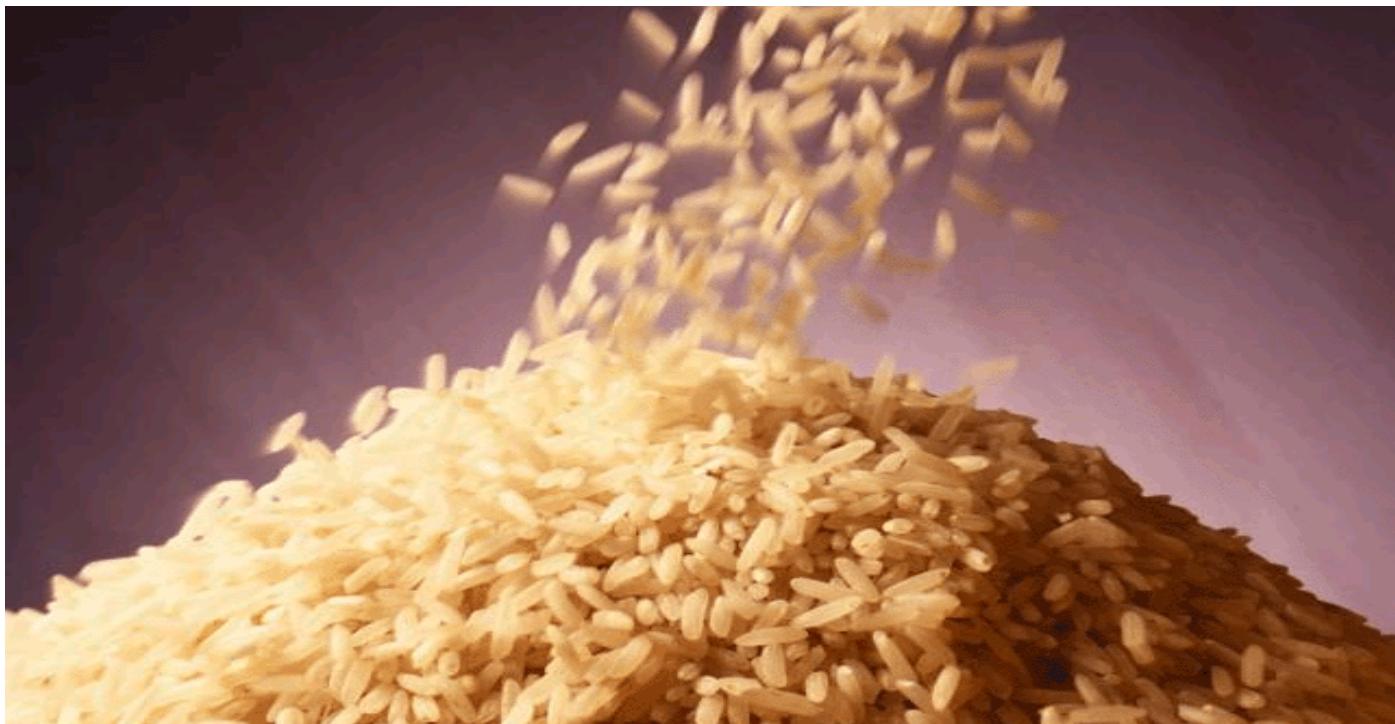
read() yordamı

- *InputStream* sınıfı soyut bir sınıfır ve bu sebepten dolayı **read()** yordamının kullanılışını görmek için *InputStream* soyut sınıfından türemiş olan *FileInputStream* sınıfı kullanılacaktır.



DosyaOkumaBir.java

read() yordamı



```
read(byte[] b) ve  
read(byte[] b, int off, int len)
```

Irmak içerisindeki gelen verileri tek tek okumak yerine, bloklar (tamponlama-buffering) halinde okuyabiliriz...



DosyaOkumaIki.java



DosyaOkumaUc.java



FilterInputStream Sınıfı

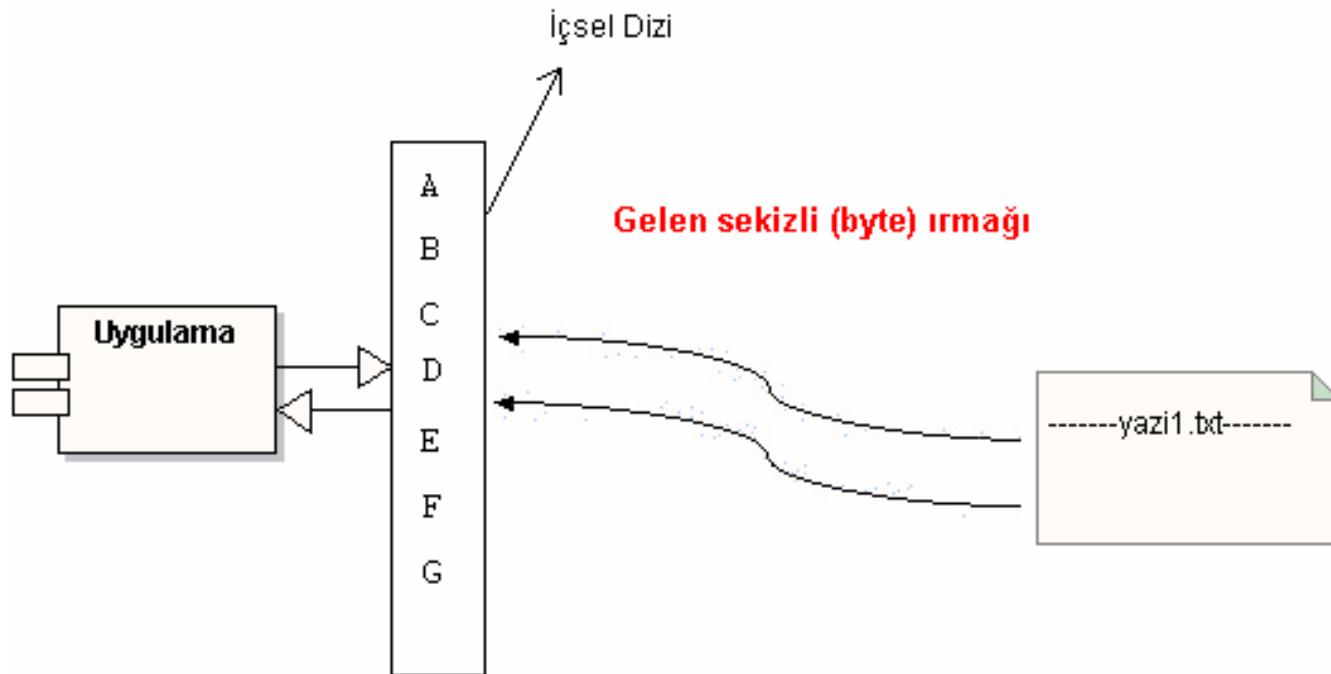
- *FilterInputStream* sınıfı *InputStream* soyut sınıfından türemiştir.
- *FilterInputStream* sınıfının ve bu sınıftan (*FilterInputStream*) türemiş diğer alt sınıfların amacı, kaynaktan gelen sekizli (byte) ırmağına ekstra özellikler katmaktır.
- Örneğin gelen ırmağın tamponlanarak performansın artırılması gibi.

BufferedInputStream Sınıfı- I

- *BufferedInputStream* sınıfı, *FilterInputStream* sınıfından türemiştir.
- *BufferedInputStream* nesnesinin içerisinde İçsel bir dizi bulunur.
- İçsel bir diziden kasıt edilen, fiziksel dosya veya soket gibi bir kaynaktan okunan sekizli verileri ilk olarak bu içsel dizinin içerisinde yerleştirilmesidir.

BufferedInputStream Sınıfı - II

▶ *BufferedOrnekBir.java*



PushbackInputStream Sınıfı

- *PushbackInputStream* sınıfı, *FilterInputStream* sınıfından türemiştir.
- Bu sınıfı kullanarak sekizli (*byte*) ırmağı içerisinde okunmuş olan bir sekizliyi (*byte*) tekrardan okumak mümkündür.



PushbackOrnekBir.java

Standart Okuma

- Kullanıcı ile uygulama arasında etkileşimi nasıl sağlanabilir?



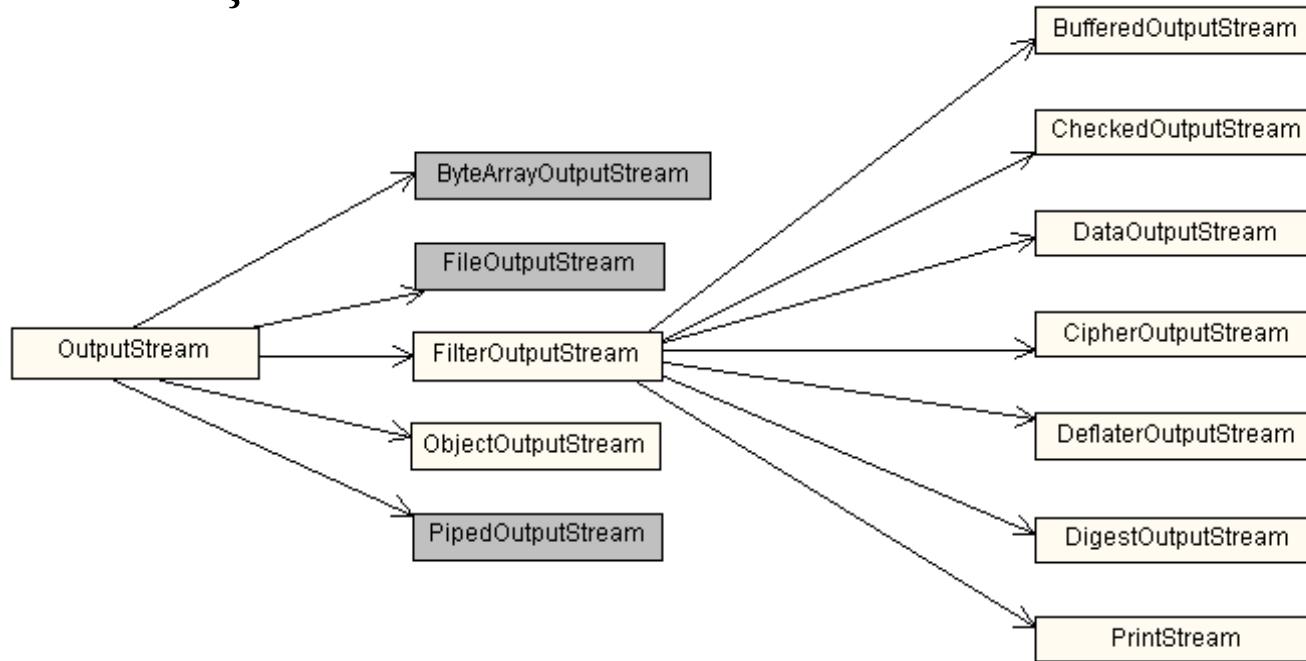
Etkilesim.java

OutputStream Soyut (*Abstract*) Sınıfı

- Şu ana kadar bir kaynaktan → uygulamamıza verilerin nasıl alınıp işlendiğini inceledik
- Şimdi sıra uygulamadan → kaynağa doğru (ör: dosya) veri yazdırma işlemlerinde...

OutputStream soyut (*Abstract*) sınıfı ve bu soyut sınıfından türemiş alt sınıflar

- Bağlantı kurup yazma işlemleri için tasarlanmış sınıflar
- Giden sekizli (*byte*) ırmaklarına özellik katmak için tasarlanmış sınıflar.



OutputStream sınıfına ait yordamlar

close()	Kaynak ile uygulama arasında kurulmuş olan bağlantıyı kapatır.
flush()	Tamponlanmış (<i>buffered</i>) olan verileri giden sekizli (<i>byte</i>) ırmağına yazdırın yordam.
write(byte[] b)	Giden sekizli (<i>bayt</i>) ırmağı içeresine belirtilen sekizli (<i>byte</i>) tipindeki diziyi aktarır.
write(byte[] b, int off, int len)	Giden sekizli (<i>bayt</i>) ırmağı içeresine belirtilen parametrelere göre yazma yapan yordam. <ul style="list-style-type: none">•byte[] b: Yazılacak olan veri dizisi.•int off: Yazılacak olan veri dizisinin kaçinci elemanından başlanması gerektiğini belirten parametre.•int len: Veri dizisinden kaç elemanın <u>giden sekizli ırmağına</u> yazılıacağını belirten parametre.
write(int b)	Belirtilen veriyi <u>giden bayt ırmağına</u> yazan yordam. Bu yordamın türemiş alt sınıflar tarafından iptal edilmesi gereklidir.

write(int b) yordamı

- Sekizli (*byte*) ırmağının yönü uygulamadan → kaynağa doğrudur.



DosyaYazmaBir.java

`write(byte[] b) ve write(byte[] b, int off, int len)`



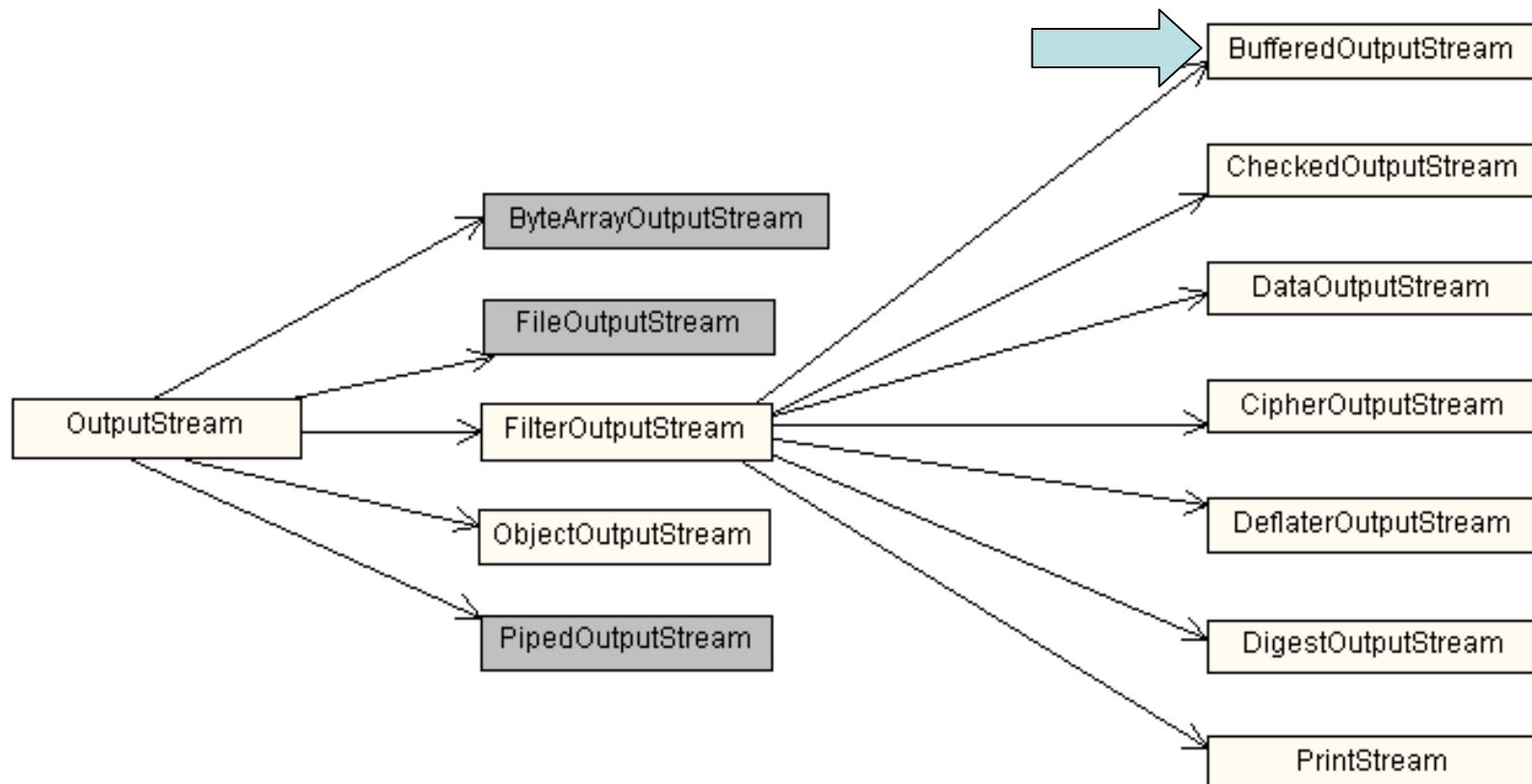
DosyaYazmaIki.java



DosyaYazmaUc.java

FilterOutputStream sınıfı

FilterOutputStream ve bu sınıfından türemiş alt sınıfların görevi, giden sekizli (bayt) ırmağı yazılacak olan sekizli (byte) verilerine yeni özellikler katmaktır.



BufferedOutputStream sınıfı - I

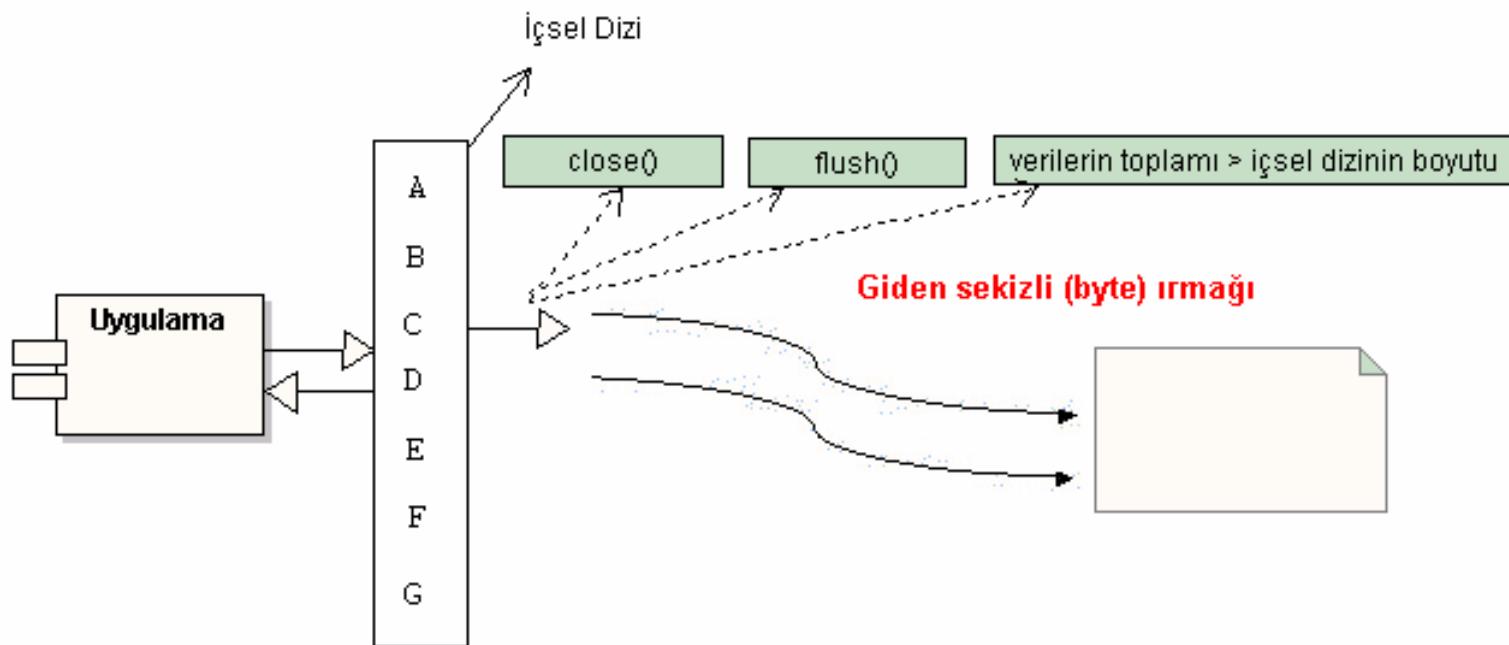
- Bu sınıfı kullanarak, giden sekizli (*bayt*) ırmağına yazılacak olan *byte*'ları önceden tamponlayabiliriz.



BufferYazmaBir.java

BufferedOutputStream sınıfı - II

Sekizli(*byte*) bilgileri ne zaman giden sekizli ırmağına yazılırlar ?



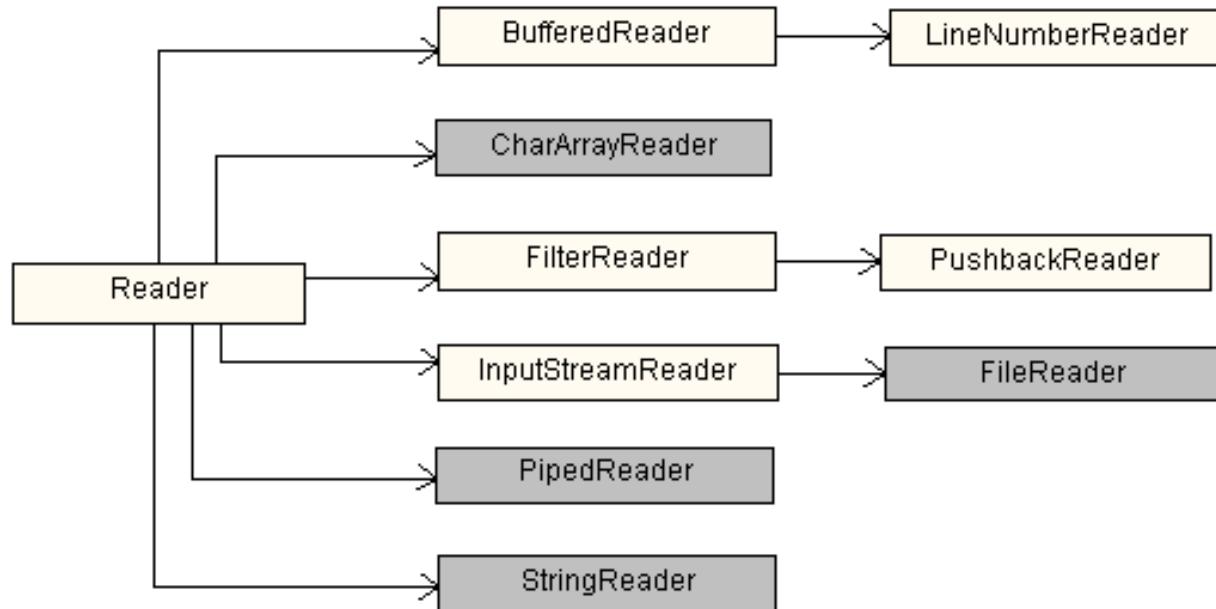
Önemli Noktalar

- JDK 1.1 'den önce sadece 8-bit'lik sekizli (*bayt*) ırmakları destekleniyordu.
- 16 bit'lik Unicode sistemi JDK 1.1 ile birlikte gelmiştir.
- Oysa; sekizli (*bayt*) ırmakları sadece ISO8859-1 karakter kümesini (*charset*) destekler.
- Karakter ırmakları Unicode sistemini destekler ve Unicode sayesinde uluslararasılaşma (i18n = *internationalization*) daha kolay sağlanır.

Karakter İrmakları (*Character Streams*)

- Karakter ırmakları üzerinde işlem yapabilen sınıfların en tepesinde iki soyut sınıf (*abstract class*) bulunur.
 - *Reader*
 - *Writer*
- *java.io* paketinin altında eğer bir sınıfın ismi *Reader* ile veya *Writer* ile bitiyorsa bu sınıf karakter ırmakları üzerinde işlem yapması için tasarlanmıştır.

Reader soyut sınıfı



- Gelen karakter ırmağının açılımını bir kez daha yaparsak:
 - Gelen ifadesi, kaynaktan (dosya, soket... gibi) uygulamaya doğru bir hareket olduğunu belirtir.
 - Karakter ifadesi, uygulama ile kaynak arasında kurulmuş olan bağlantının içerisinde temel (*primitive*) *char* tipinde verilerin akacağına işaretettir.
 - Irmak ise uygulama ile kaynak arasında kurulan bağlantıyı ifade eder.

Reader soyut sınıfına ait yordamlar.

close()	Uygulama ile kaynak arasında <u>kurulmuş</u> olan bağlantıyı kapatır.
mark(int okumalimiti)	İçsel dizi kullanılarak tamponlanmış (<i>buffered</i>) olan veriler içerisinde istenilen pozisyonu işaret koyar.
markSupported()	<u>Gelen karakter ırmağı</u> için mark() ve reset() yordamlarının çalışabilirliğini test eden yordam.
read()	<u>Gelen karakter ırmağı</u> içerisinde tek bir karakter (<i>char</i>) okuyan yordam.
read(char[] cbuf)	<u>Gelen karakter ırmağı</u> içerisinde belirtilen karakter (<i>char</i>) dizisi kadar okuma yapan yordam.
read(char[] cbuf, int off, int len)	<u>Gelen karakter ırmağı</u> içerisinde belirtilen parametrelere göre okuma yapan yordam.
ready()	<u>Gelen karakter ırmağının</u> okunabilecek durumda olup olmadığını belirten yordam.
reset()	İşaretlenen pozisyonu geri döndüren yordam.
skip(int n)	<u>Gelen karakter ırmağı</u> içerisinde <u>n</u> baytlık veriyi atlar

InputStreamReader Sınıfı (Köprü)

- Bu sınıfın rolü köprü görevi görmektedir.
- Bu sınıf, gelen sekizli (*bayt*) ırmakları ile gelen karakter ırmakları arasında köprü vazifesi görür.

```
import java.io.*;
public class ISROrnek {

    public static void main(String args[]) throws IOException {
        InputStream in = System.in;
        InputStreamReader unicode = new InputStreamReader( in );
        //...
    }
}
```

FileReader sınıfı

- *FileReader* sınıfı *InputStreamReader* sınıfından türemiştir. Bu sınıfın rolü dosyadaki verileri karakter ırmağı şeklinde uygulamaya taşımaktır.

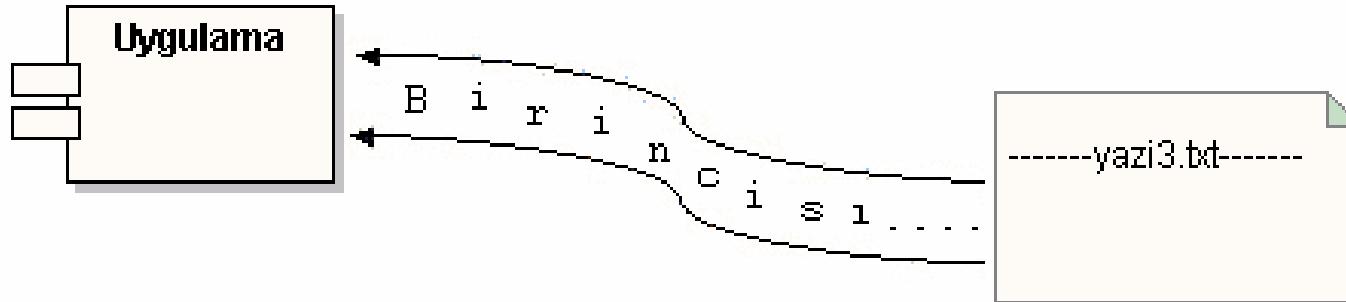


FileReaderOrnekBir.java

Birinci sınıfıtken,
İkinci sınıfıtken, Üçüncü sınıfıtken,
Dördüncü sınıfıtken, Beşinci sınıfıtken,
Altıncı sınıfıtken, Yedinci sınıfıtken,
Sekizinci sınıfıtken, Dokuzuncu sınıfıtken,
Onuncu sınıfıtken, Onbirinci sınıfıtken,
Onikinci sınıfıtken, Onuçüncü sınıfıtken,
Onordüncü sınıfıtken, Onbeşinci sınıfıtken
Onaltıncı sınıfıtken.....

yazi3.txt

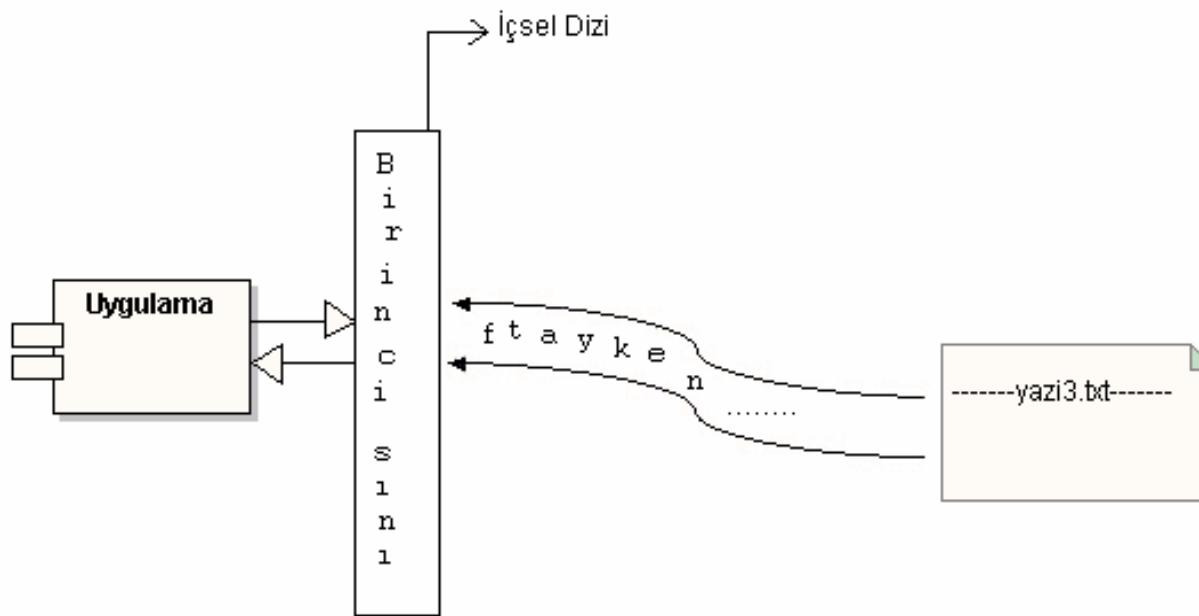
Ekran Çıktısı



- Tek tek okuma i\$lemi --
encoding:Cp1254 == ISO8859-9
toplamSatirSonu:10
- Bloklar halinde okuma i\$lemi –
encoding:Cp1254 == ISO8859-9
toplamSatirSonu:10
- Blok halinde belirtilen çerçevede okuma i\$lemi -
encoding:Cp1254 == ISO8859-9
toplamSatirSonu:10

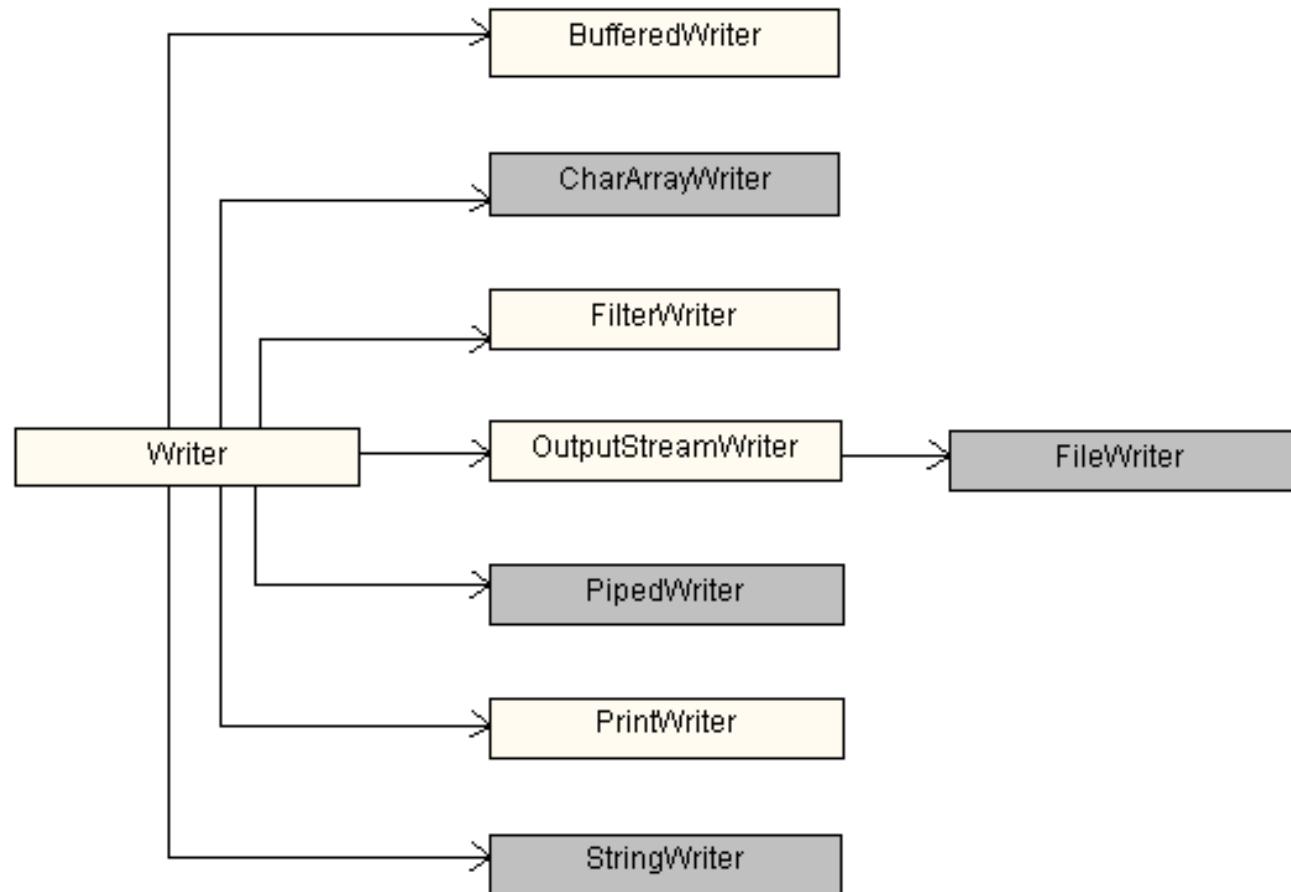
BufferedReader Sınıfı

- *BufferedReader* sınıfını kullanmak performansı ciddi bir şekilde artırmaktadır.



BufferedReaderOrnekBir.java

Writer soyut sınıfından türemiş diğer alt sınıflara ait şema



Writer soyut sınıfına ait yordamlar

close()	Kaynak ile uygulama arasında kurulmuş olan bağlantıyı kapatır. Kapama işleminin hemen öncesinde veriler <u>giden karakter ırmağına</u> yazılır.
flush()	Tamponlanmış (<i>buffered</i>) olan verileri giden karakter ırmağına yazdırın yordam.
write(char cbuf[])	<u>Giden karakter ırmağı</u> içeresine belirtilen karakter (char) dizisini yazar
write(char cbuf[], int off, int len)	Giden karakter ırmağı içeresine belirtilen parametrelere göre yazma yapan yordam. <ul style="list-style-type: none">• char [] cbuf: Yazılacak olan veri dizisi.• int off: Yazılacak olan veri dizisinin kaçinci elemanından başlanması gerektiğini belirten parametre.• int len: Veri dizisinden kaç elemanın giden karakter ırmağına yazılacağını belirten parametre.
write(int c)	Belirtilen karakteri giden karakter ırmağına yazan yordam.
write(String str)	String nesnesini <i>giden karakter ırmağına</i> yazan yordam.
write(String str, int off, int len)	String nesnesinin belirtilen kısımlarını <u>giden karakter ırmağına</u> yazan yordam.

OutputStreamWriter ve FileWriter Sınıfı

- *FileWriter* sınıfı, bölgesel ayarları kullanarak dosyaya yazmaktadır.



FileWriterOrnek.java

Önemli Nokta

- *UnicodeBulucu.java*; bir *String* nesnesinin veya tek tek karakterlerin Unicode karşılıklarını bulması için tasarlanmıştır.
- Unicode nedir ?



UnicodeBulucu.java

<u>Karakter</u>	<u>Java'daki Unicode karşılığı</u>
A	\u0041
g	\u011F

- Bu sınıfı kullanarak verileri giden karakter ırmagına aktarmamız mümkündür.
- Bu sınıfın içerisinde sekizli (*byte*) tipindeki verileri giden karakter ırmagına aktaran bir yordam(*method*) yoktur.



PrintWriterOrnek.java

BufferedWriter Sınıfı

- Bu sınıfın rolü, karakter verilerini giden karakter ırmağına aktarılmalarından evvel tamponlayarak performansın artırılmasını sağlamaktır.
- *FileWriter* ve *BufferedWriter* nesnelerini beraber kullanmak performans açısından daha iyi bir sonuç verecektir.



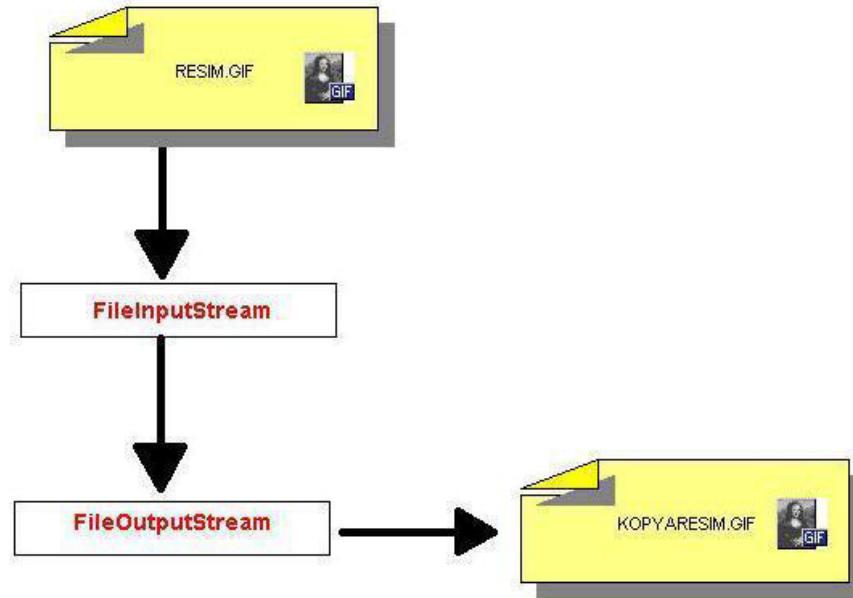
BufferedWriterOrnek.java

Performans - I

- Performans bölümünde 4 adet örnek uygulama incelenecektir.
- Bu uygulamaların yaptıkları işler hep aynı olacak, dosyaları kopyalamak...
- Kopyalanması için seçilen dosya ise 361K boyutundaki bir GIF dosyası olsun...



Test1.java

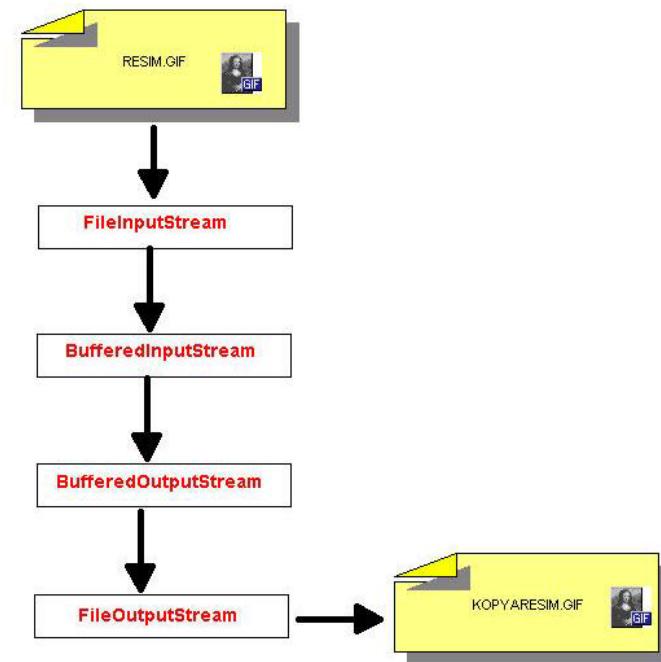


Performans - II

- Kopyalanması için seçilen dosya yine 361K boyutundaki bir GIF dosyası olsun...
- Bu yeni yaklaşımımızda *BufferedInputStream* ve *BufferedOutputStream* sınıfları kullanılacaktır; bu yaklaşım acaba performansı nasıl etkileyeyecek?



Test2.java



- Her zaman *BufferedInputStream* ve *BufferedOutputStream* sınıflarını kullanmak zorunda değiliz.
- Bunun yerine kendi oluşturduğumuz dizileri, tampon vazifesi görmesi için kullanabiliriz.



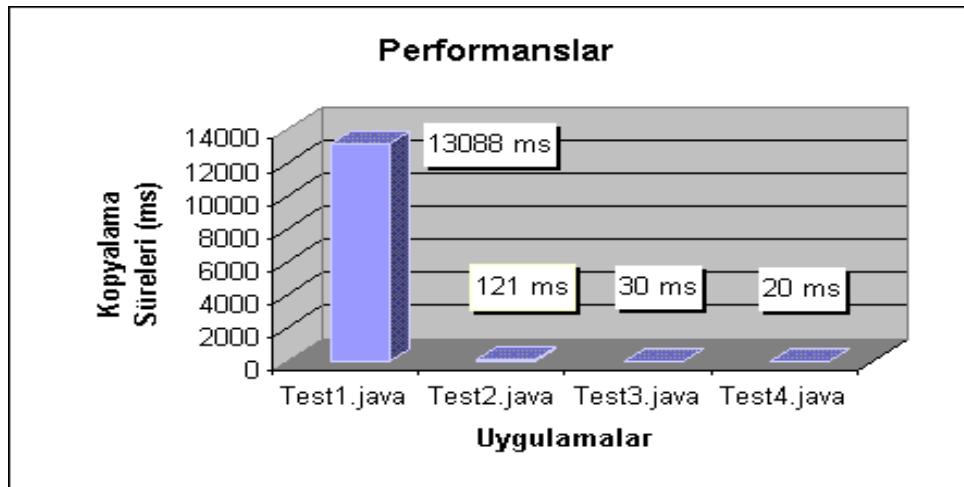
Test3.java



Test4.java

Performans Sonuçları

- *Test1.java* uygulamasını kullanırsak, toplam boyutu 361K olan bir GIF dosyanın kopyalanması için gereken süre 13088 ms
- *Test2.java* uygulamasını kullanırsak, toplam boyutu 361K olan bir GIF dosyanın kopyalanması için gereken süre 121 ms
- *Test3.java* uygulamasını kullanırsak, toplam boyutu 361K olan bir GIF dosyanın kopyalanması için gereken süre 30 ms
- *Test4.java* uygulamasını kullanırsak, toplam boyutu 361K olan bir GIF dosyanın kopyalanması için gereken süre 20 ms



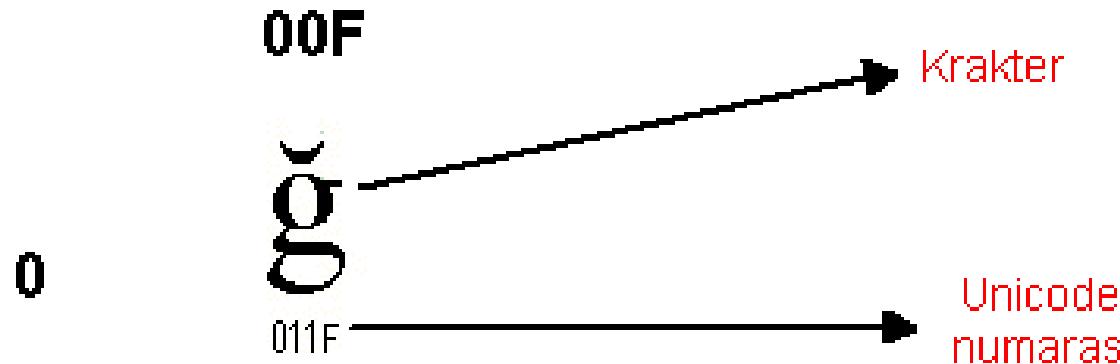
- *Unicode*, platformu ve programlama dili ne olursa olsun dünya üzerindeki herbir karaktere ayrı ayrı tekil numaralar atanarak geliştirilmiş bir standarttır.
- Java programlama dilinde kullanılan *String* nesnelerinin içerisindeki veriler *Unicode* sistemine uygun olarak tutulur.
- Uluslararasılaştırma = *Internationalization* (i18n)

		000	001	002	003	004	005	006	007	008	009	00A	00B	00C	00D	00E	00F
0	@	P	0	@	P	`	p	Ä	Ê	†	°	À	Ğ	à	ğ		
1	A	!	1	A	Q	a	q	Å	ë	i	±	Á	Ñ	á	ñ		
2	B	"	2	B	R	b	r	Ç	Í	c	²	Â	Ò	â	ò		
3	C	#	3	C	S	c	s	É	í	£	³	Ã	Ó	ã	ó		
4	D	\$	4	D	T	d	t	Ä	î	o	'	Ä	Ô	ä	ô		
5	E	%	5	E	U	e	u	Ö	ï	¥	µ	Å	Õ	å	õ		
6	F	&	6	F	V	f	v	Ü	ñ		¶	Æ	Ö	æ	ö		
7	G	'	7	G	W	g	w	á	ó	§	.	Ç	×	ç	÷		
8	H	(8	H	X	h	x	à	ò	..	,	È	Ø	è	ø		
9	I)	9	I	Y	i	y	â	ô	©	¹	É	Ù	é	ù		
A	:	J	*	:	J	Z	j	z	ää	ö	a	o	Ê	Ú	ê	ú	
B	;	K	+	;	K	[k	{	ää	ö	«	»	Ë	Û	ë	û	
C	<	L	,	<	L	\	l		å	ú	¬	½	Ì	Ü	ì	ü	
D	=	M	-	=	M	l	m	}	ç	ü	R	½	Í	İ	í	ı	
E	>	N	.	>	N	^	n	~	é	û	®	¾	Î	Ş	î	ş	
F	?	O	/	?	O	_	o	Q	ë	ü	—	¿	İ	ß	ï	ÿ	

ISO8859-9 Tablosu

Detaylı Bakış

- ISO8859-9 tablosundaki "g" karakteri yakından incelenirse;



ISO8859-9 tablosundaki koordinatı = 00F0



000	001	002	003	004	005	006	007	008	009	00A	00B	00C	00D	00E	00F	
0	@	P	0	@	P	`	p	Ä	ê	†	°	À	Đ	à	đ	
1	A	!	1	A	Q	a	q	Å	ë	i	±	Á	Ñ	á	ñ	
2	B	"	2	B	R	b	r	Ç	í	c	²	Â	Ò	â	ò	
3	C	#	3	C	S	c	s	É	ì	£	³	Ã	Ó	ã	ó	
4	D	\$	4	D	T	d	t	Ä	î	o	'	Ä	Ô	ä	ô	
5	E	%	5	E	U	e	u	Ö	ï	¥	µ	Å	Õ	å	õ	
6	F	&	6	F	V	f	v	Ü	ñ	-	¶	Æ	Ö	æ	ö	
7	G	'	7	G	W	g	w	á	ó	§	·	Ç	×	ç	÷	
8	H	(8	H	X	h	x	à	ò	..	,	È	Ø	è	ø	
9	I)	9	I	Y	i	y	â	ô	©	¹	É	Ù	é	ù	
A	:	J	*	:	Z	j	z	ä	ö	a	o	Ê	Ú	ê	ú	
B	;	K	+	;	K	[k	{	ã	õ	«	»	Ë	Û	ë	û
C	<	L	,	<	L	\	l		å	ú	¬	¼	Ì	Ü	ì	ü
D	=	M	-	=	M]	m	}	ç	ù	R	½	Í	Ý	í	ý
E	>	N	.	>	N	^	n	~	é	û	®	¾	Î	Þ	î	þ
F	?	O	/	?	O	_	o	Q	è	ü	-	¿	İ	ß	ï	ÿ

ISO8859-1 Tablosu

000	001	002	003	004	005	006	007	008	009	00A	00B	00C	00D	00E	00F	
0	@	P	0	@	P	`	p	Ä	ê	†	º	Ā	Ð	ā	đ	
1	A	!	1	A	Q	a	q	Å	ë	À	à	Á	Ñ	á	ñ	
2	B	"	2	B	R	b	r	Ç	Í	K	্	Â	Õ	â	õ	
3	C	#	3	C	S	c	s	É	ì	R	্	Ã	Ķ	ã	ķ	
4	D	\$	4	D	T	d	t	Ä	î	○	'	Ä	Ô	ä	ô	
5	E	%	5	E	U	e	u	Ö	ï	Í	í	Å	Õ	å	õ	
6	F	&	6	F	V	f	v	Ü	ñ	L	ł	Æ	Ö	æ	ö	
7	G	'	7	G	W	g	w	á	ó	§	্	ł	×	í	÷	
8	H	(8	H	X	h	x	à	ò	”	,	Č	Ø	č	ø	
9	I)	9	I	Y	i	y	â	ô	š	š	É	Ù	é	ù	
A	:	J	*	:	Z	j	z	ää	ö	Ē	ē	E	Ú	ę	ú	
B	;	K	+	;	K	k	{	ää	ö	G	ǵ	Ë	Û	ë	û	
C	<	L	,	<	L	\	l	å	ú	F	Ń	Ü	é	ü		
D	=	M	-	=	M	l	m	ç	ù	R	N	Í	Ü	í	ü	
E	>	N	.	>	N	^	n	~	é	û	Ž	ž	Î	Ù	î	ú
F	?	O	/	?	O	_	o	Q	è	ü	—	ŋ	Í	ß	í	·

ISO8859-4 Tablosu

Örnek



UnicodeTest.java

File Sınıfı

- *File* sınıfı fiziksel dosyaları temsil ederler.

FileTestBir.java

```
import java.io.*;
public class FileTestBir {
    public static void main(String[] args) throws IOException {
        File dosya = new File("Test1.txt");
    }
}
```

Soru : Bu uygulama çalıştırılırsa *Test1.txt* dosyası olusur mu ?

Dönüşüm (*Serialization*)

- Dönüşüm bir nesnenin durum bilgisini saklanabilecek ve taşınabilecek şekilde dönüştürme işlemidir.
 - Dönüşüm işlemini kullanarak nesneleri ağ üzerinden başka bir makinaya gönderebilir
 - Sabit diske kayıt edilebilir.
- Dönüşüm sayesinde nesnelerin ömürleri, uygulamanın ömrüne bağlı olmaktan çıkar.



DonusumTest1.java

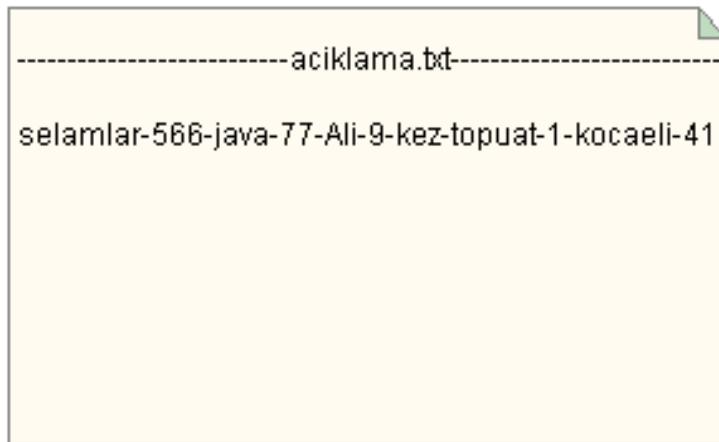
Bölümleyiciler (*Tokenizer*)

- Bölümleyiciler, kendilerine gelen karakterleri böлerek parçalara ayırır.
- Bu parçaların tipi sayı veya harf olabilir.
 - *StreamTokenizer* sınıfı
 - *StringTokenizer* sınıfı

StreamTokenizer sınıfı



Bolumleyici.java



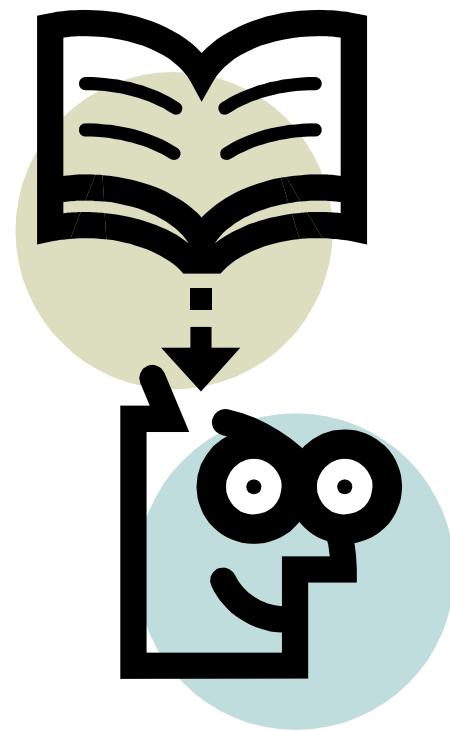
StringTokenizer sınıfı

- Bu sınıf, *java.util* paketinin içerisinde bulunmaktadır.
- Bu sınıfın *StreamTokenizer* sınıfından farkı ise, bölümlediği parçaları *String* tipinde geri döndürmesidir.

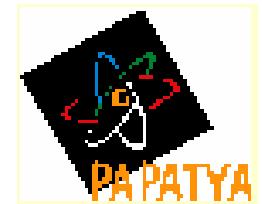
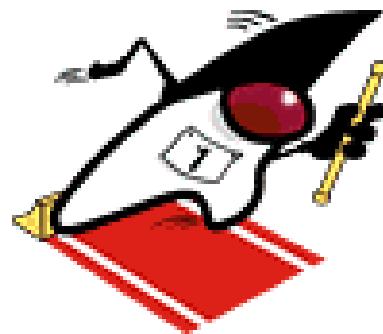


StringTokenizerOrnekBir.java

Sorular ...



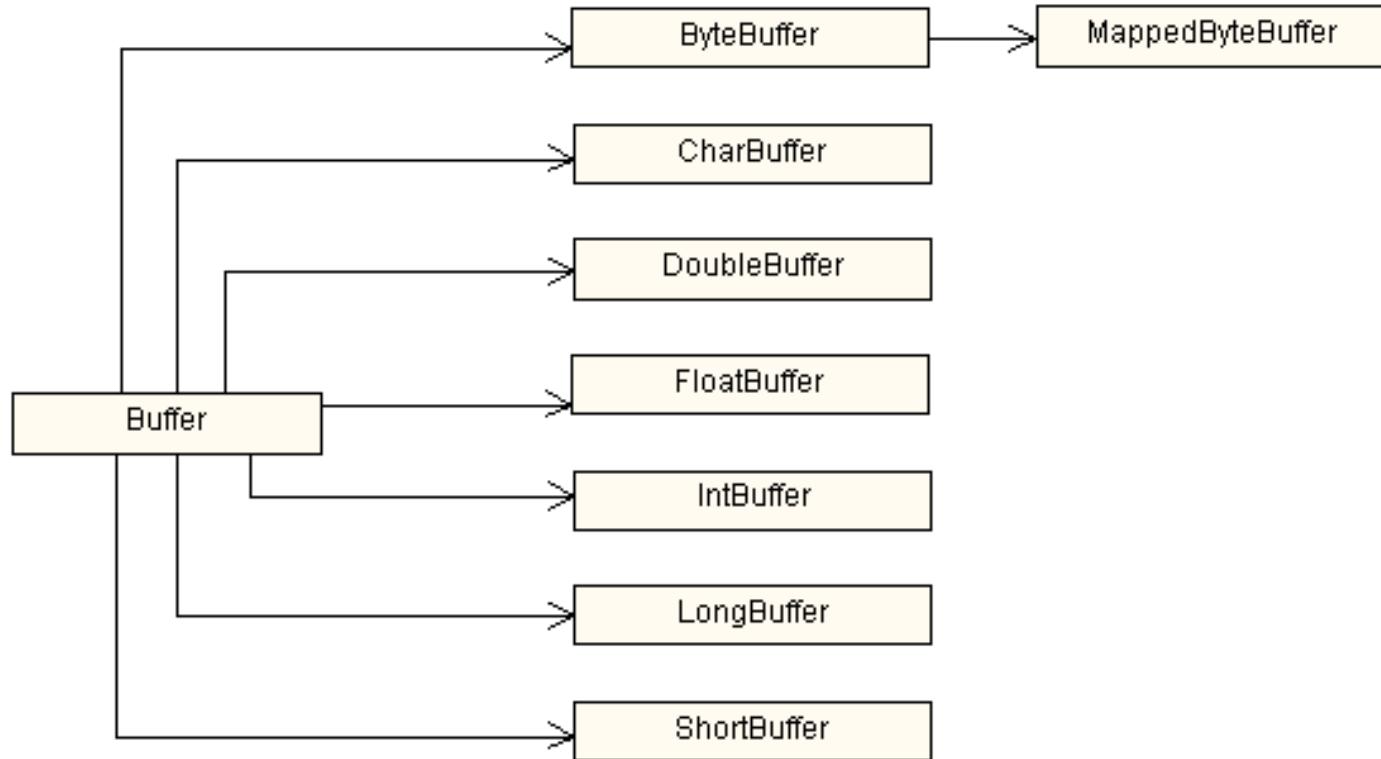
Yeni G/Ç (*new I/O*)



YENİ G/Ç (NEW I/O)

- Yeni G/Ç (*nio*) paketi içerisindeki sınıfları kullanarak dosya işlemleri (okuma/yazma) ve ağ üzerinden okuma/yazma işlemleri gerçekleştirebilir.
- Buradaki fark, dosya işlemlerinin her zaman engellemeli (*blocking*) ama ağ işlemlerinin ise opsiyonel olmasıdır.
- Yani ağ işlemleri isteğe bağlı olarak engellemeli ve engellemesiz (*non-blocking*) olabilir.

Tamponlar (*Buffers*)



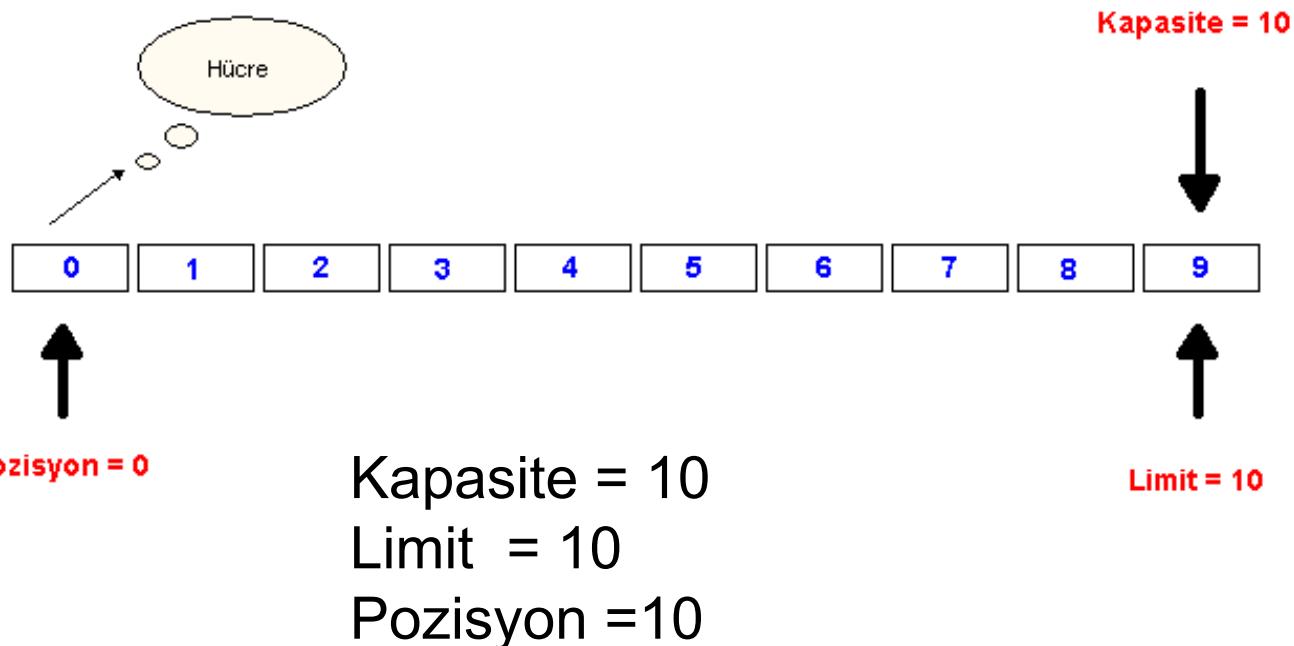
ByteBuffer Sınıfı

- **Tamponun kapasitesi** : Kapasite parametresinin değeri bir kere ve en başta belirtilir ve bir daha değiştirilemez. Bu parametre, tamponun kaç adet eleman alacağını belirler.
- **Tamponun limiti** : Tampon içerisindeki okunamayacak veya yazılamayacak olan elemanın indeksini belirtir. Limit, negatif veya tamponun kapasitesinden büyük olamaz.
- **Tamponun pozisyonu** : Tampon içerisindeki okunabilecek veya yazılabilecek olan elemanın indeksini belirtir. Pozisyon, negatif veya limit değerinden büyük olamaz.

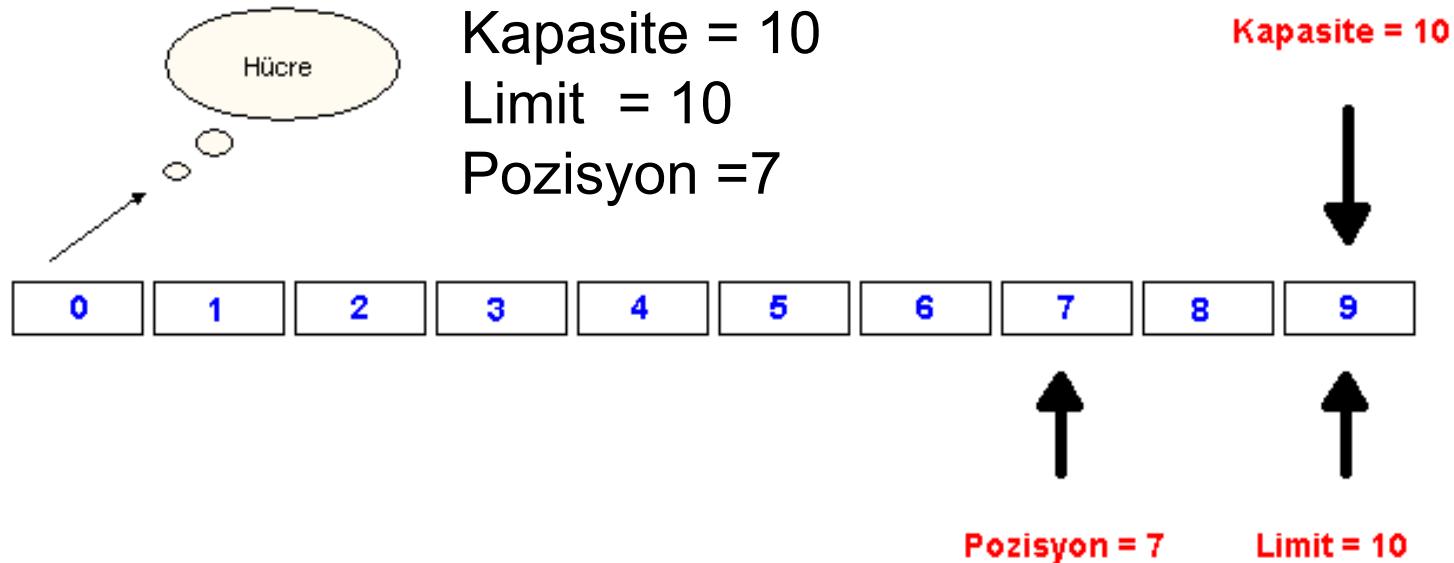


NIO Test Bir.java

Yeni oluşturulan ByteBuffer nesnesi



İçersine veri atılmış ByteBuffer nesnesi



Pozisyon değerinin 7 olmasındaki sebep,
okunabilecek ve yazılabilecek olan hücrenin
indeksinin 7 olmasından kaynaklanır.



NIOCTestIki.java

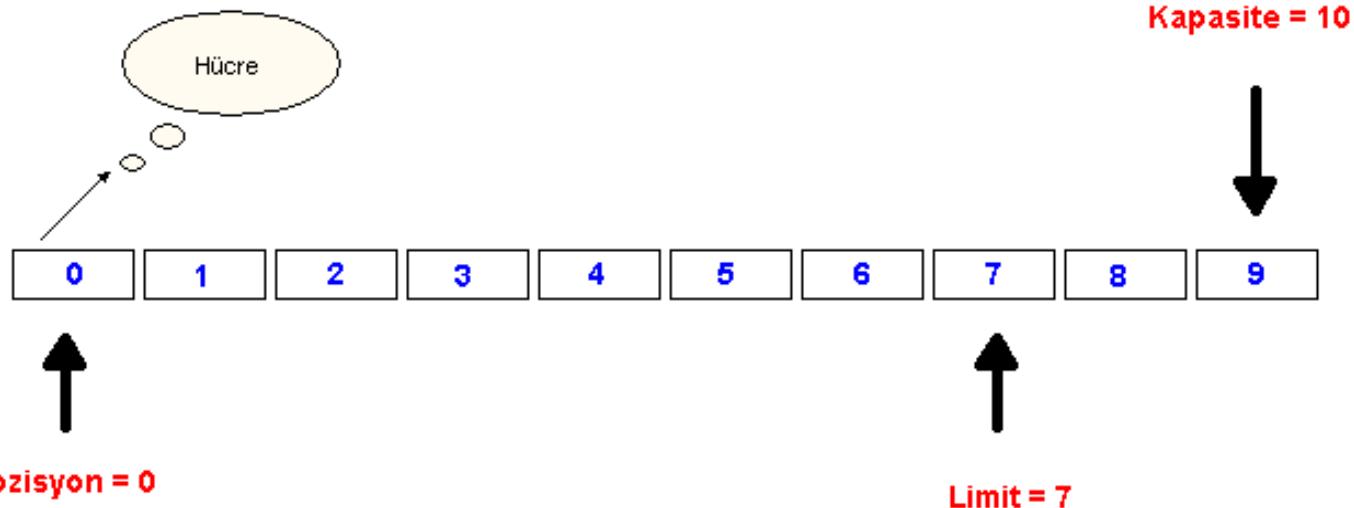
hasRemaining(), flip(), get() Yordamları

- **hasRemaining()** yordamı, mevcut pozisyon ile limit arasındaki eleman sayısını bildirir.
- **flip()** yordamı çağrıldığında, *ByteBuffer* nesnesinde tanımlı olan pozisyonun değeri sıfıra eşitlenir. Limit değeri okunamayacak veya yazılamayacak olan son elemanın üzerine getirilir ve biraz ileride göreceğimiz **mark()** yordamının kullanılmasıyla işaretlenmiş olan yer silinir.
- **get()** yordamı, belirtilen elemanı okur.



NIOTestUc.java

ByteBuffer nesnesinin, **flip()** yordamı çağrıldıktan sonraki hali



- Limit değeri neden 7 oldu?
- Daha doğrusu limit diye bir değer niye var?

Relatif ve Mutlak Operasyonlar (Relative and absolute operations)

- Verileri içeri yerleştirme veya içerisindeki verileri çekme işlemleri *Buffer* soyut sınıfından türemiş her alt sınıf için 2 kategori bulunur.
 - **Relatif Operasyonlar**
 - **Mutlak Operasyonlar**

Relatif Operasyonlar

- Relatif operasyon kullanılarak içeri atılan her veri için mevcut pozisyon değeri bir artar.
- Atılan verileri çekmek için kullanılan **get()** yordamını çağrımadan evvel, pozisyon değerinin sıfırlanacağını düşünürsek (**flip()** yordamını kullanarak), içерiden çekilen her eleman için pozisyonun değeri bir artacaktır.
- **put()** yordamını kullanırken limit aşımı oluşursa *BufferOverflowException* istisnası oluşur.
- **get()** yordamını kullanırken bir limit aşımı oluşursa da *BufferUnderflowException* istisnası oluşur.



RelatifOrnek.java

Mutlak Operasyonlar

- Mutlak operasyon kullanılarak içeri atılan verilerde veya mevcut verileri çekiş işlemlerinde belirli bir indeks numarası kullanılır.
- Mutlak operasyonlarda mevcut pozisyonun değerinde bir artış olmaz.



MutlakOrnek.java

- İlkel tipte olan bir dizi, *java.nio* paketi içerisindeki tampon sınıflarını kullanarak sarmalanabilir.
- Bu işlem için **wrap()** yordamını kullanmamız yeterli olacaktır.



Sarmaliyici.java

Doğrusal ve Doğrusal Olmayan Tamponlar (*Direct and non-direct buffers*)

- Doğrusal ve doğrusal olmayan tamponlar sadece *ByteBuffer* sınıfına ait olan özelliklerdir.
 - Doğrusal *ByteBuffer* nesnesi oluşturmak için **allocateDirect()** yordamını çağrırmak yeterlidir.
 - Doğrusal olmayan *ByteBuffer* nesnesi oluşturmak için **allocate()** yordamını çağrırmak yeterlidir.



DogrusalOrnekBir.java



DogrusalOlmayanOrnekBir.java

ByteOrder Sınıfı

- Bir verinin tipi sekizli (*byte*) tipinden büyükse, bu verinin bölünüp sekizli tipindeki verilere dönüştürülerek saklanması gereklidir.
- Örneğin ilkel int tipindeki bir veri, 4 sekizlik (32 bit) verilere bölünerek saklanır veya ilkel short tipindeki bir veri 2 sekizlik (16 bit) verilere dönüştürülerek saklanır.
- Verilerin bu saklanma işlemi CPU mimarilerinin geçmişten gelen farklılıklarından dolayı değişiklik gösterebilir.

- Bir tipteki tampon nesnesini başka bir tipteki tampon nesnesine çevirebiliriz.
- Örneğin *ByteBuffer* nesnesini, *DoubleBuffer* nesnesine kolaylıkla çevirebiliriz.



CeviriciOrnek.java

Tampon Nesneler ve Diziler

- Tampon nesnelerin, arka planda bildiğimiz dizileri kullanması için doğrusal olmayan (*non-direct*) bir şekilde oluşturulmuş olmaları gereklidir.



DiziOrnek.java

- *java.nio* paketinin altındaki tampon sınıflarını kullanarak yapılan işaretleme ve işaretlenen yere geri dönme işlemleri gayet kolaydır.

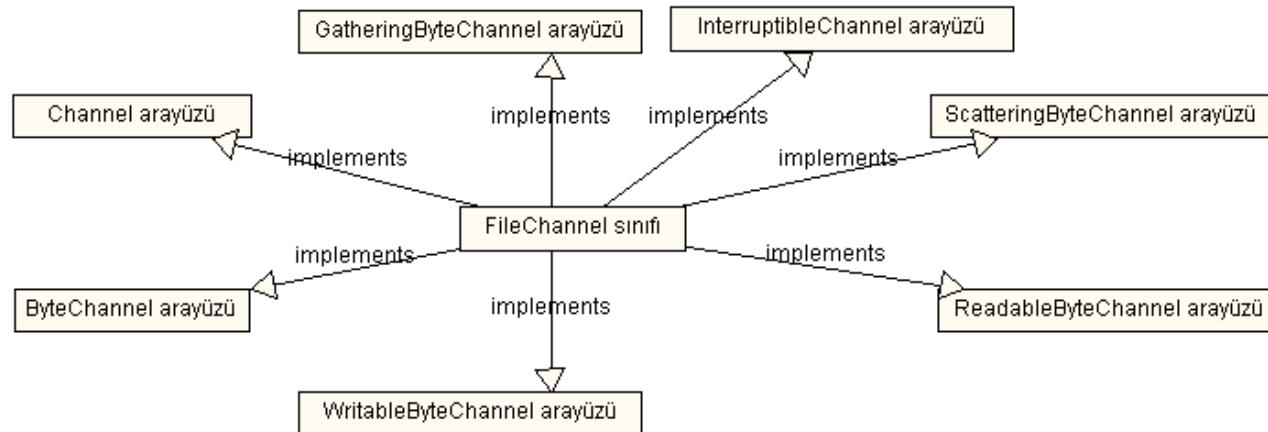


IsaretlemeOrnek.java

- Kanal özelliğini kullanarak, uygun bir cihaz, bir dosya, ağ üzerinden bir soket veya uygun bir uygulama ile bağlantı kurabilirler.
- Java programlama diline versiyon 1.4'de katılmış olan bu özellik, geçen bölümde incelediğimiz *InputStream* ve *OutputStream* soyut sınıflarının (bunlardan türemiş olan alt sınıfların) sağladıkları özelliklerin pabuçlarını biraz da olsa dama atmaktadır.
- Bu bölümümüzde sadece *FileChannel* sınıfını inceleyeceğiz.

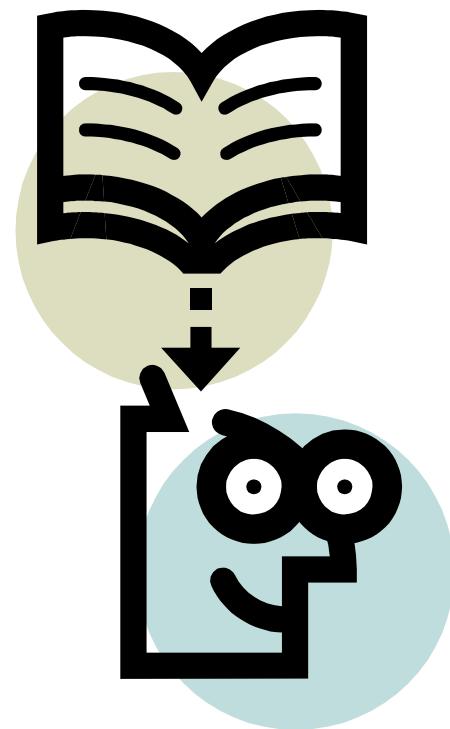
FileChannel Sınıfı

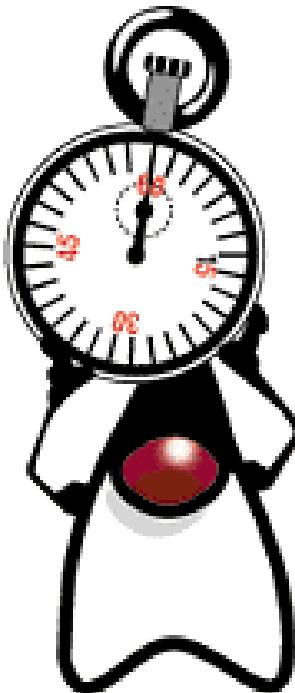
- Bu sınıf sayesinde mevcut fiziksel bir dosya ile bağlantı kurup okuma, yazma, eşleme (*mapping*) ve dosya yönetim işlemlerini gerçekleştirebiliriz.



ChannelOrnekBir.java

Sorular ...





İş Parçacıkları

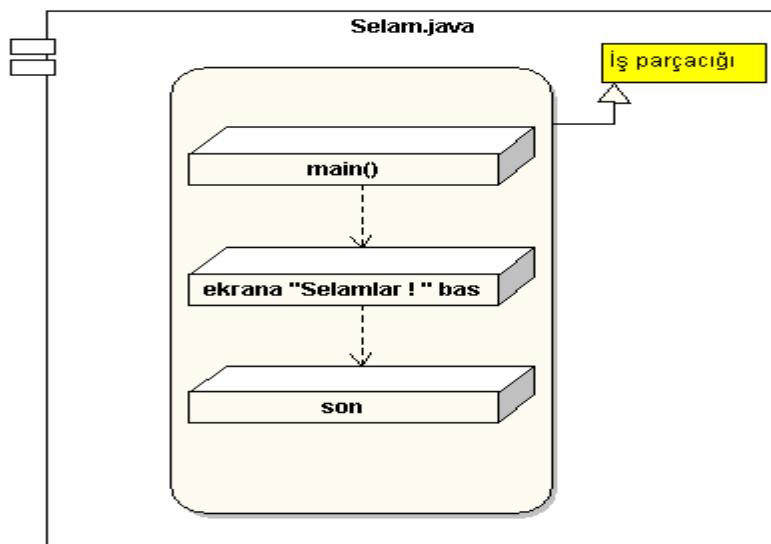
(Threads)

**Bir elin nesi var iki elin sesi var
-Atasözü-**



İŞ PARÇACIKLARI (THREADS)

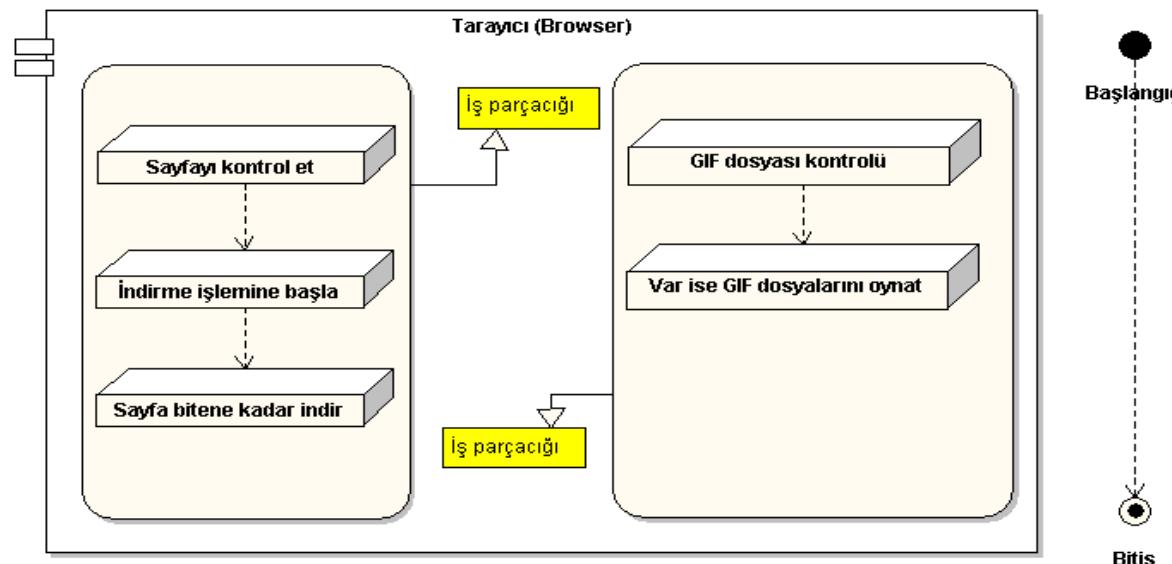
- Geçen bölümlerde yapılan uygulama örnekleri hep sıralıydı.
- Program başlar, belli bir yolu izleyerek işlemler yapar ve biterdi.



Selam.java

Çoklu İş Parçacıklarına ne zaman ihtiyaç duyulur ?

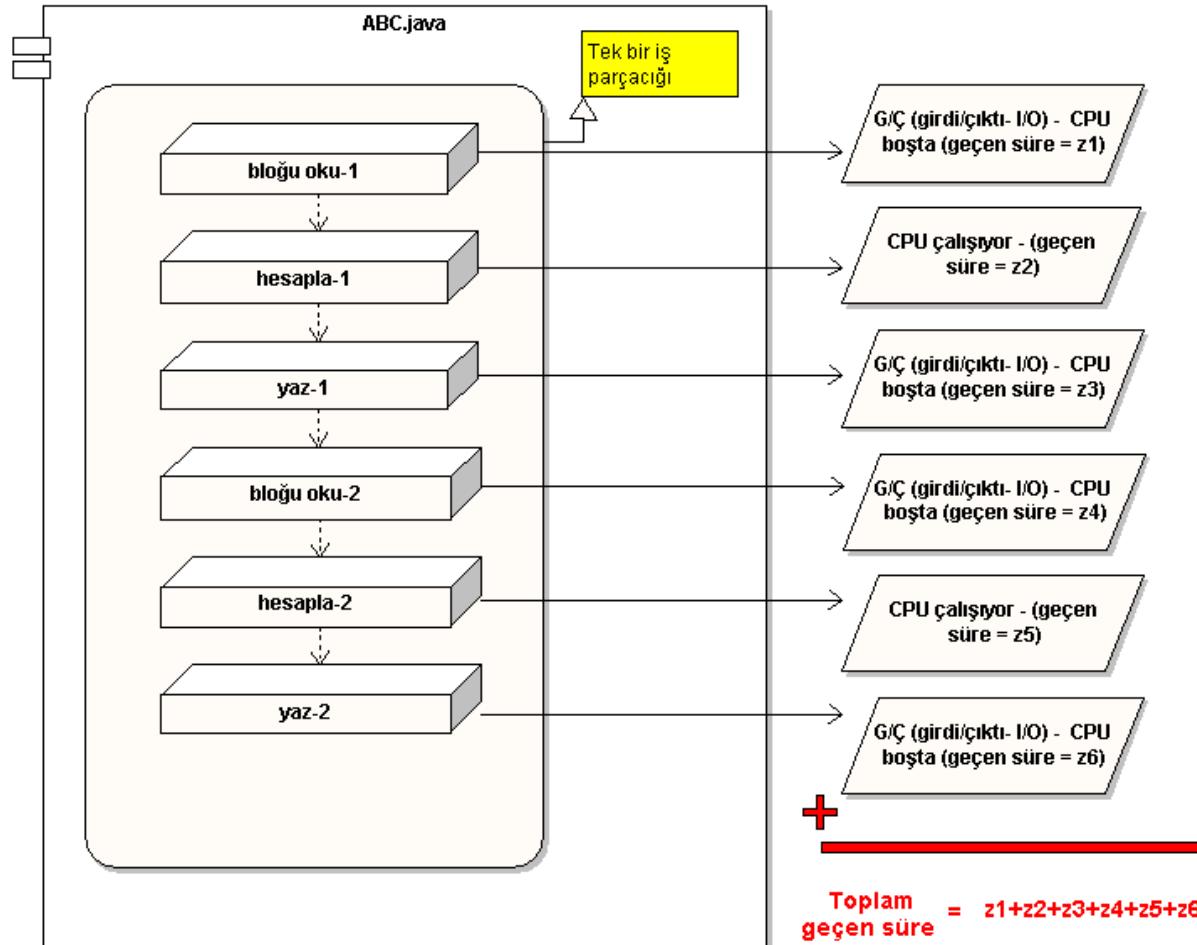
- Bu durumlara en iyi örnek tarayıcılardır (*browser*).
 - İstenilen sayfanın indirilmesi için bir iş parçacığı
 - İndirilmiş olan GIF dosyalarını oynatmak için bir iş parçacığı



Sohbet – I

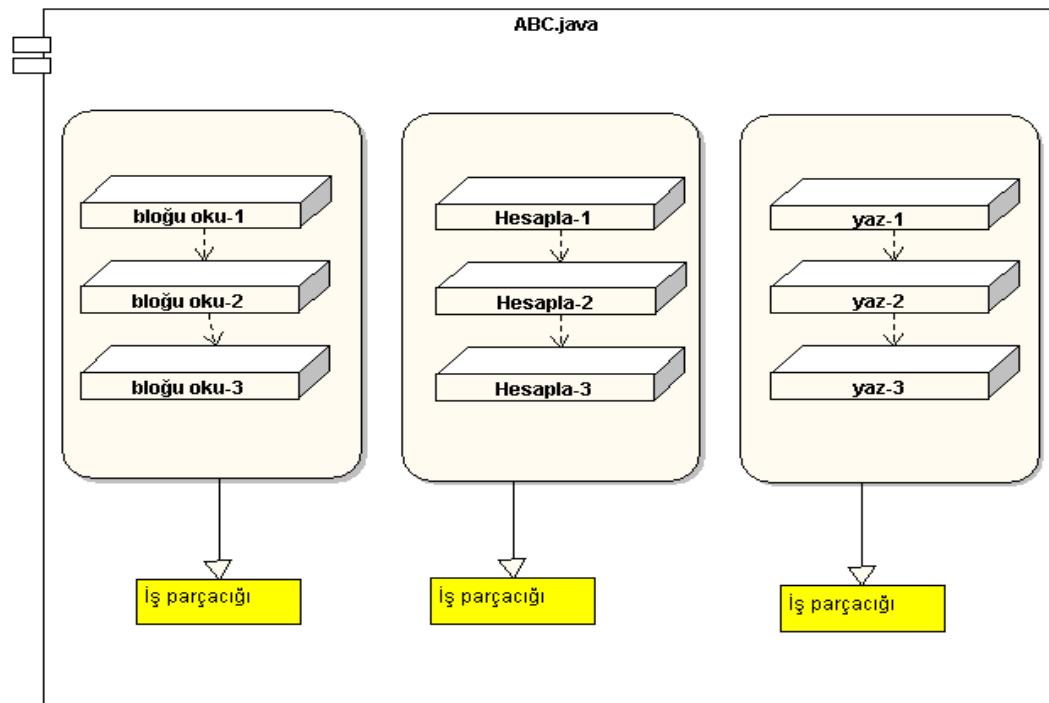
- Şimdi öyle bir uygulama düşünelim ki:
 - Bu uygulama bir dosyadan okuma yapsın,
 - Okuduğu veri üzerinde hesaplama yapıp,
 - Hesaplamanın sonucunu başka bir dosyaya yazsın.
- Burada kaç işleminden bahsediyoruz?
 1. Dosyadan okuma yapma (G/Ç)
 2. Okunan veri üzerinde hesaplama yapma (CPU çalışıyor)
 3. Hesaplama sonucunu başka bir dosyaya yazma (G/Ç)

Sohbet – II (Tek bir iş parçacığından oluşan uygulamanın aşamaları)



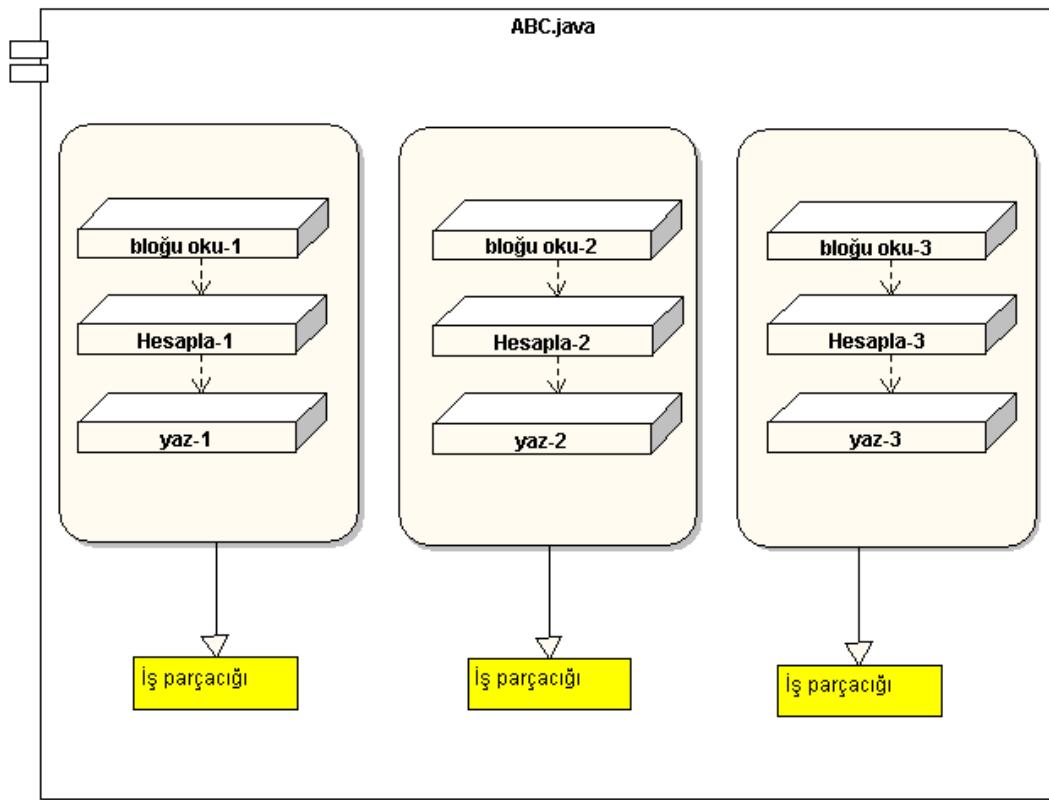
Sohbet – III (1. Tasarım)

- Bu uygulamanın gerçeklestirmesi gereken 3 ana işlem (okuma-hesaplama-yazma) olduğunu biliyoruz.
- Bu üç işlemi tek bir iş parçacığında yapmaktansa, üç ayrı iş parçacığı içerisinde yaparsak sonuç nasıl değişir?



Sohbet – IV (2. Tasarım)

- İş parçacıkları ile aşağıdaki gibi bir tasarım da yapılabilir.



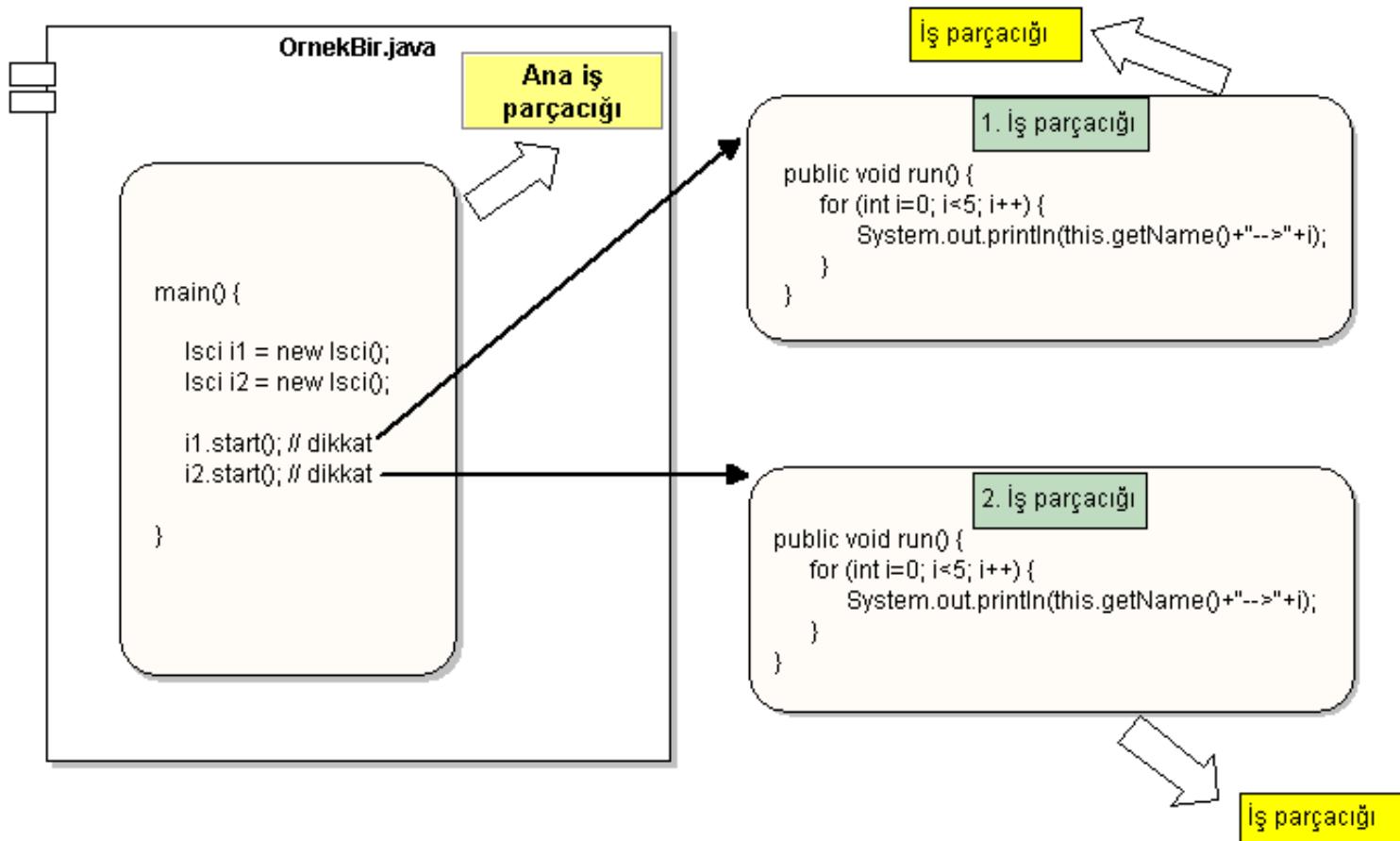
İş Parçacıklarının Başlatılması

- Tek başına çalışabilen (*standalone*) uygulamaların başlangıç yeri statik **main()** yordamı (*methods*) olduğunu daha evvelden belirtmiştim.
- Uygulama çalışmaya başladığında, ana iş parçacığı oluşturulup olayların akışı başlatılır.
- Java programlama dili ile yazdığımız uygulamaların içerisinde çoklu iş parçacıklarını kullanmak için *java.lang.Thread* sınıfını veya *java.lang.Runnable* arayüzüünü kullanmamız gereklidir.

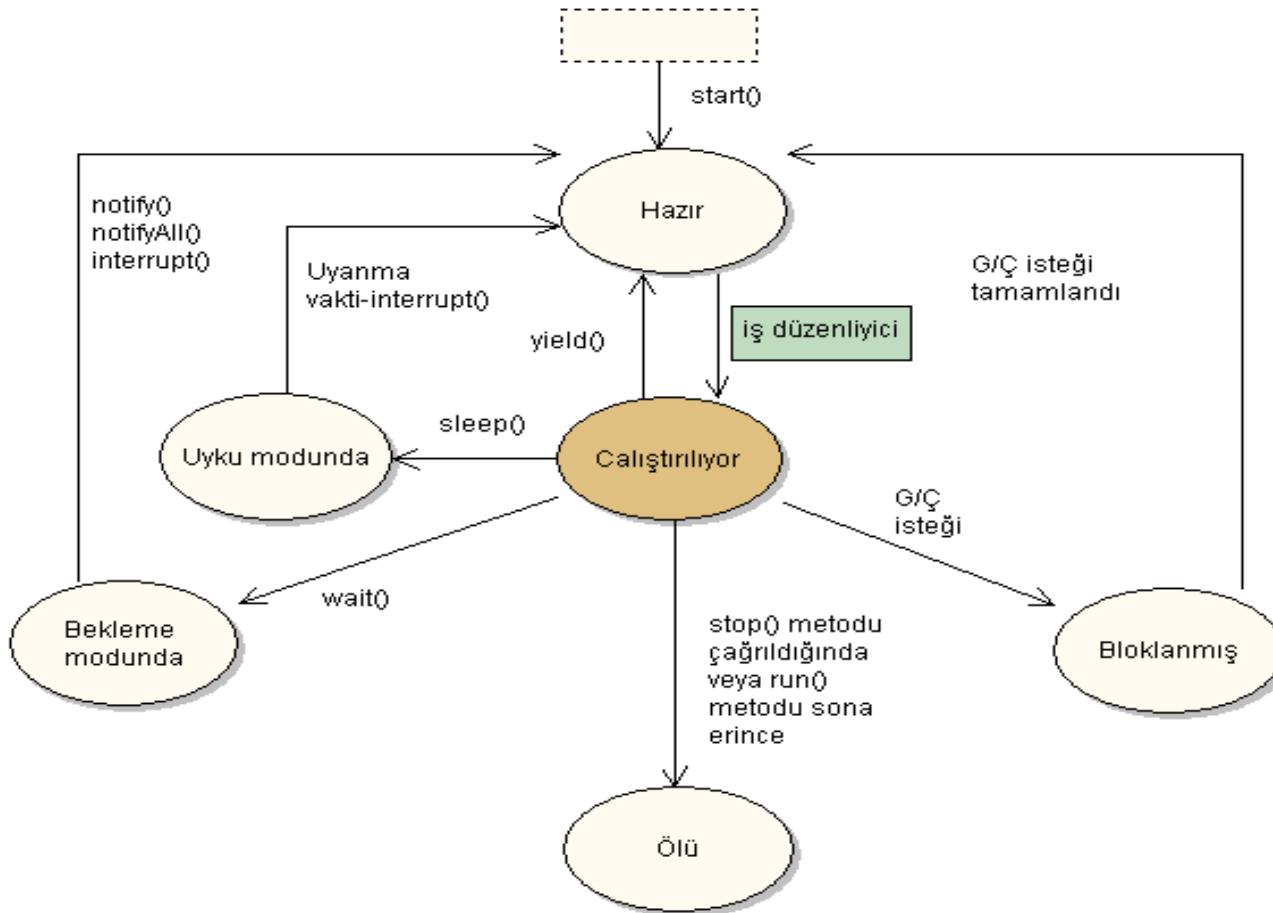


OrnekBir.java

ÖrnekBir.java uygulamasının çalışması



Haller



Öncelik Sırası

- Aynı öncelik sırasına sahip olan iş parçacıkları aynı hazır durum havuzunda bulunurlar.

Thread.MIN_PRIORITY = 1

Thread.NORM_PRIORITY = 5

Thread.MAX_PRIORITY = 10



OrnekIki.java

İş Parçacıklarının Sonlandırılması

- Bir iş parçacığının sonlanması onun ölmesi anlamına gelir.
- Peki bir iş parçacığı nasıl öldürebiliriz?
 - Birinci yol ilgili iş parçacığının `stop()` yordamını çağırarak gerçekleştirilebilir ama bu tavsiye edilmeyen bir yoldur.
 - İkinci yol nasıl olabilir ?



OrnekUc.java

İş Parçacıklarının Kontrolü

- **sleep()** : Çalışan iş parçacığının belirli bir süre uyumasını sağlar. Bu statik bir yordamdır; yani bu **sleep()** yordamını çağrırmak için *java.lang.Thread* sınıfından türemiş alt bir sınıfın **new()** anahtar kelimesi ile oluşturulması gerekmez.



UykuTest.java

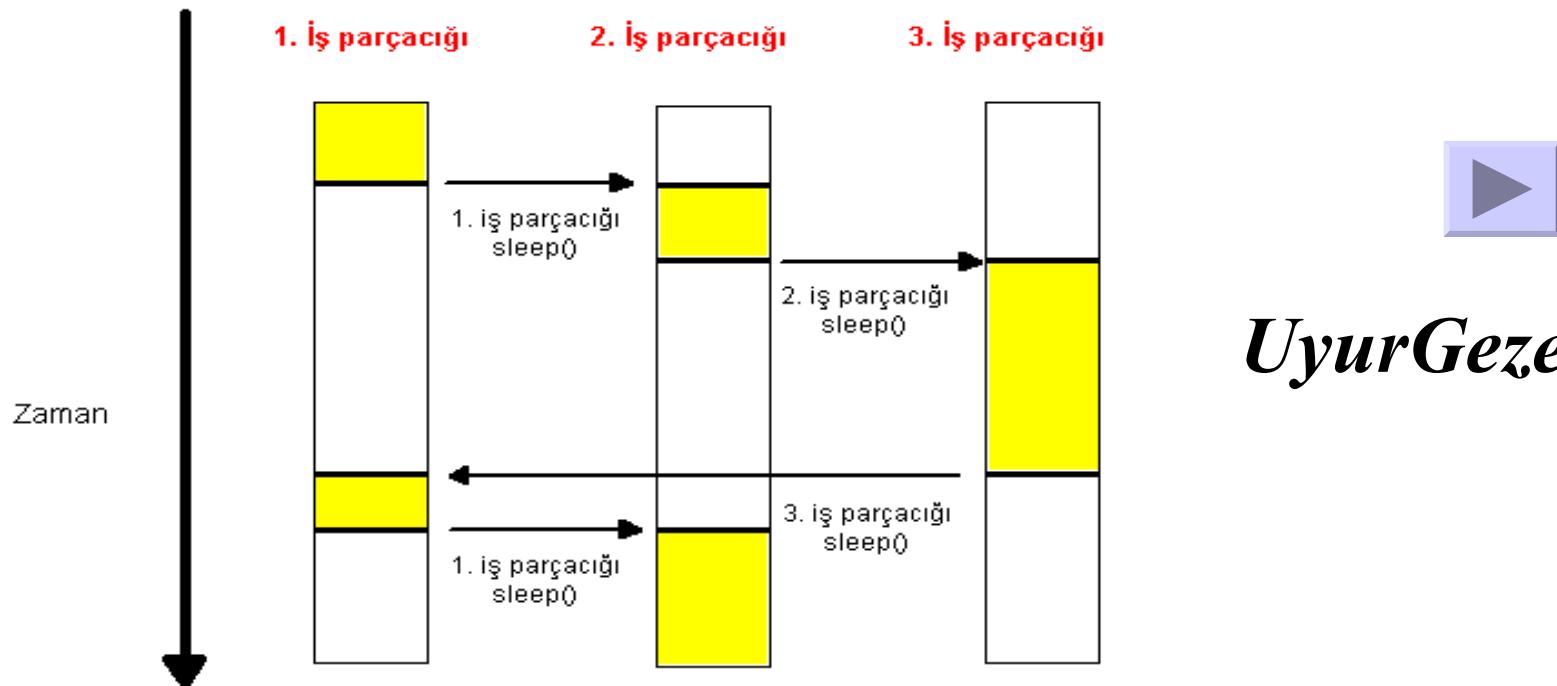
- **interrupt()** : Uyuyan bir iş parçacığını uyandırmanın yolu onu rahatsız etmektir. Bu rahatsızlık verme olayını **interrupt()** yordamını çağrıarak başarabiliriz.



UyanmaVakti.java

sleep() yordamı

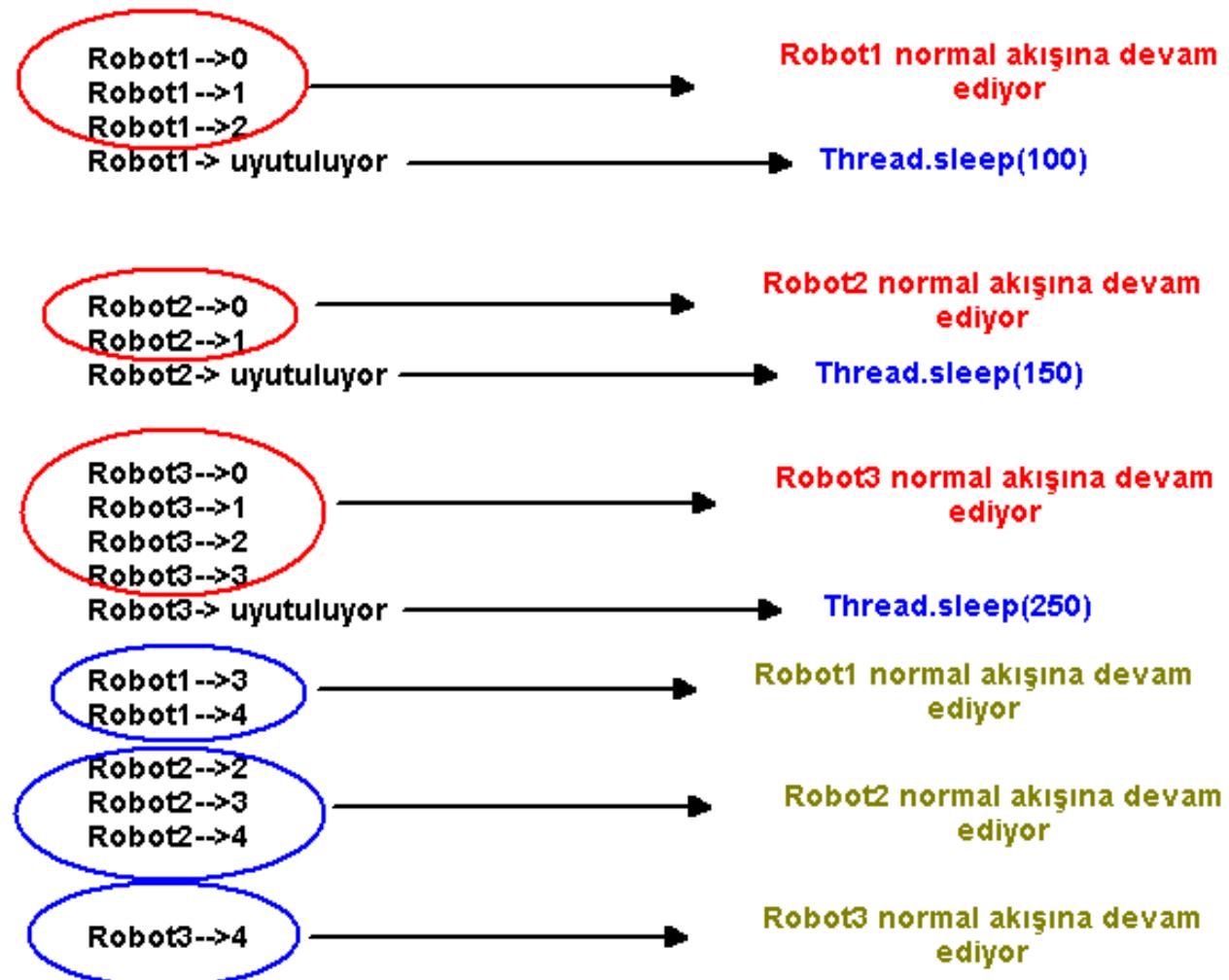
- Elimizde 3 adet iş parçacığı olduğunu ve bu üç iş parçacığının da aynı anda başlatıldıklarını hayal edelim...



UyurGezer.java

Koyu alanlar iş parçacıklarının çalışıklarını zamanı gösterir.

UyurGezer.java uygulamasının ekran çıktısı

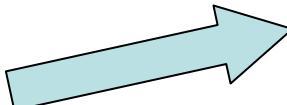


yield() Yordamı

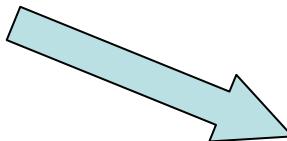
- Bir iş parçacığı çalıştırılıyor halinde iken, bu iş parçacığı ile aynı öncelik sırasına sahip başka bir iş parçacığına çalışma fırsatı vermek istiyorsak **yield()** yordamını kullanmamız gereklidir.



YieldOrnek.java



Normal bir şekilde çalışıralım...



Şimdi 9. satırındaki yorum satırını açalım, derleyelim ve çalışıralım...

join() Yordamı

- **join()** yordamı, bir iş parçacığının diğer bir iş parçacığını beklemesi için kullanılır.
- **join()** yordamının üç adaş yordamı (*overloaded*) bulunur.

join()	Belirtilen iş parçacığı bitene (ölü haline gelene kadar) kadar bekletir.
join(long milisaniye)	Belirtilen iş parçacığını, verilen milisaniye kadar bekletir.
join(long milisaniye, int nanosaniye)	Belirtilen iş parçacığını, verilen milisaniye + nano saniye kadar bekletir.



JoinTest.java

Tasarım - Thread Sınıfından Kalıtım

- Bir sınıfı ait nesneyi iş parçacığına dönüştürmek için iki tasarım modeli bulunmaktadır.
- Bunlardan ilki, şu ana kadar yaptığımız gibi ilgili sınıfı *java.lang.Thread* sınıfından türetmektir.

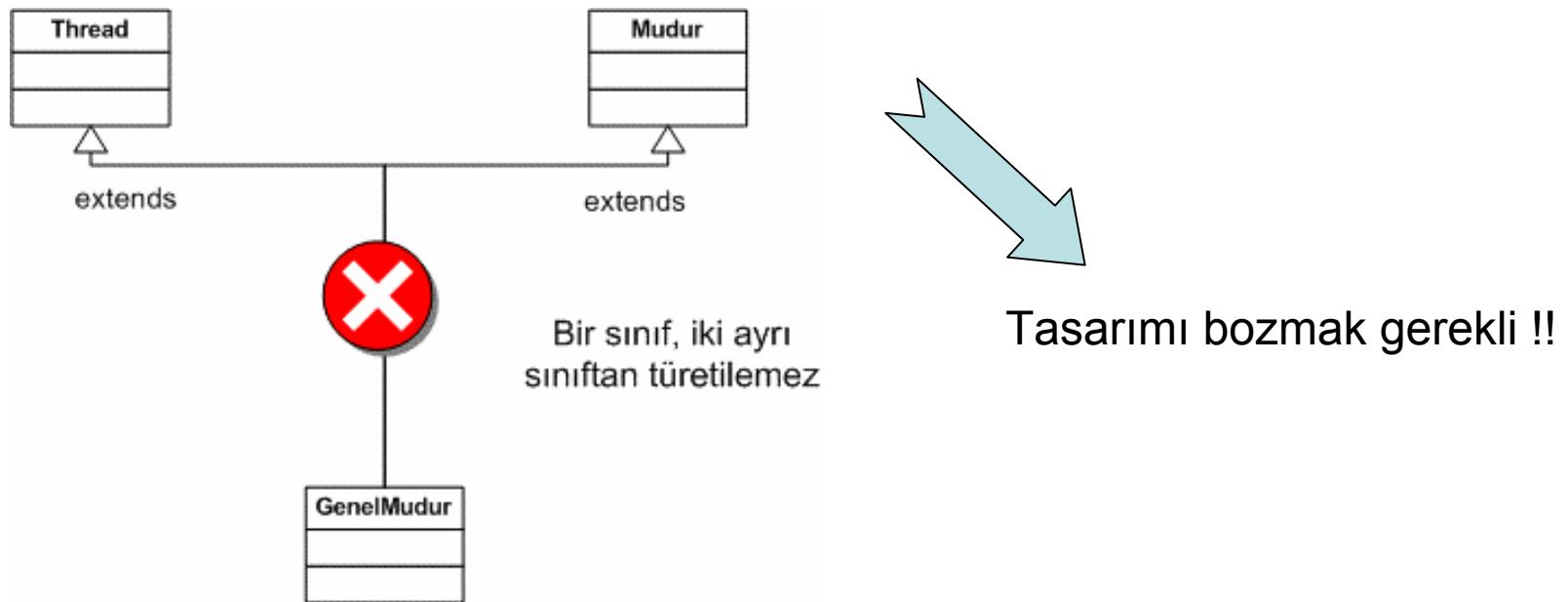
```
public class OrnekSinif extends Thread {  
  
    public void run() {  
        // ...  
    }  
}
```

Tasarım - Thread Sınıfından Kalıtımın Avantajları

- Bu tasarımın avantajı daha kolay kodlama denilebilir.
- Örneğin **run()** yordamının içerisinde **getName()** yordamını direkt çağrıabilirmiz.

Tasarım - Thread Sınıfından Kalıtımın Dezavantajları

- Java programlama dilinde bir sınıf ancak ve ancak tek bir diğer sınıfından türetilenbildiği için (*single inheritance*) bu model kullanılarak tasarlanan iş parçacıklarında belirli kısıtlamalar gündeme gelebilir.

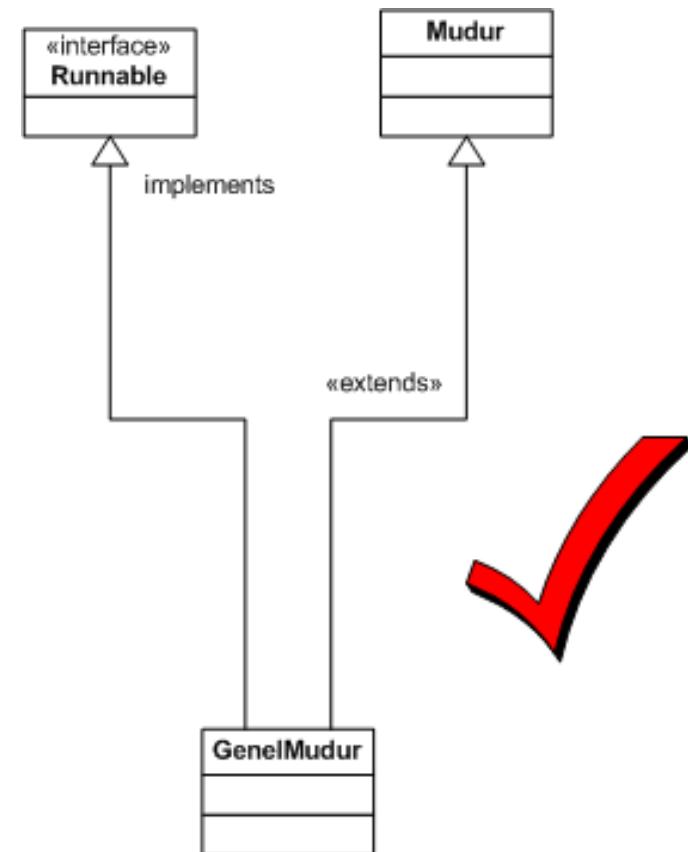


Tasarım - Runnable Arayüzü

- *Runnable* arayüzü sayesinde bir sınıfı iş parçacığına dönüştürmek mümkündür.
- *Runnable* arayüzünü kullanmanın dezavantajları olarak daha uzun kodlama denilebilir.

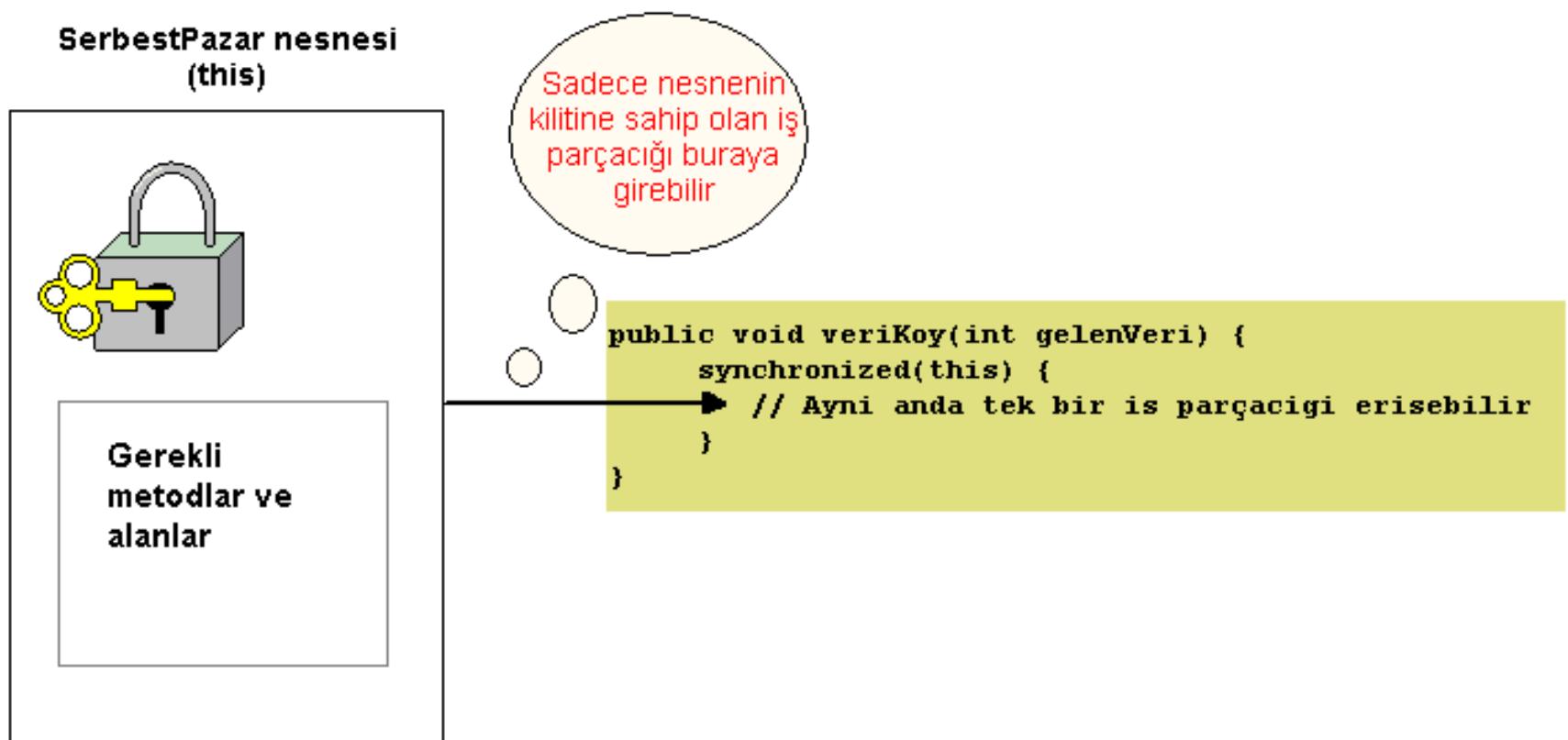


ArayuzTest1.java



- Her nesnesin kendisine ait bir kiliti bulunur.
- Bir sınıfı ait bir nesne oluşturulurken bu kilit otomatik olarak oluşur.
- Bu kiliti eline geçen iş parçacığı, kritik alan üzerinde işlem yapmaya hak kazanır.
- Kritik alan, birden fazla iş parçacığının aynı anda üzerinde işlem yapmaması gereken bölgedir.

Kritik Alan



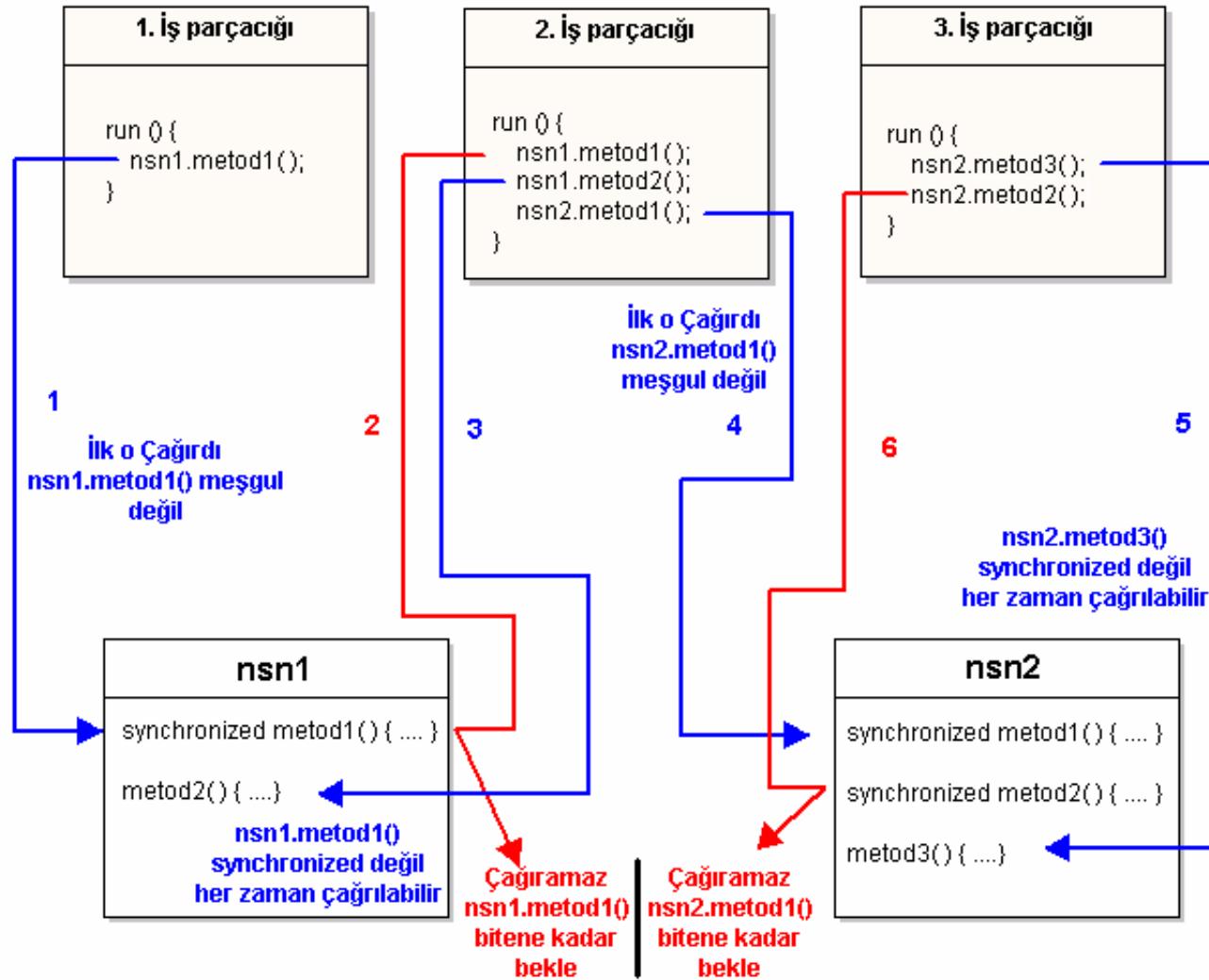
synchronized Anahtar Kelimesi - I

- Bir yordam veya yordamın içerisindeki kritik bir bölge **synchronized** anahtar kelimesi ile korunma altına alınabilir.
- Korunma altına alınmaktan kasıt, aynı anda iki veya daha fazla iş parçacığının bu kritik bölgeye veya yordamın komple kendisine erişmesini engellemektir.

```
public synchronized void veriKoy(int gelenVeri) {  
    // tum yordam koruma altında  
    // ayni anda bir tek is parcacigi erisebilir  
}
```

```
public void veriKoy(int gelenVeri) {  
    synchronized(this) {  
        // sadece belirli bir kisim koruma altında  
    }  
}
```

synchronized Anahtar Kelimesi - II



synchronized Anahtar Kelimesi - III



FotokopiMakinasi.java

wait(), notify() ve notifyAll() Yordamları

- Her nesnenin bir kilidi olduğu gibi bir de bekleme havuzu (*object's monitor*) bulunur.
- Bu bekleme havuzuna iş parçacıkları atılır -**wait()** - veya bu havuzdan dışarı çıkartılır - **notify()** / **notifyAll()** -
- Bu beş yordam (**wait()** yordamının iki de adaş yordamı bulunur), *java.lang.Object* nesnesinin içerisinde bulunur.

Semafor (Semaphore)

- Kaynağın az olduğu durumlarda bir çok iş parçacığı arasında bir düzen sağlamak gereklidir.



Semaphore.java



SemaphoreTest.java

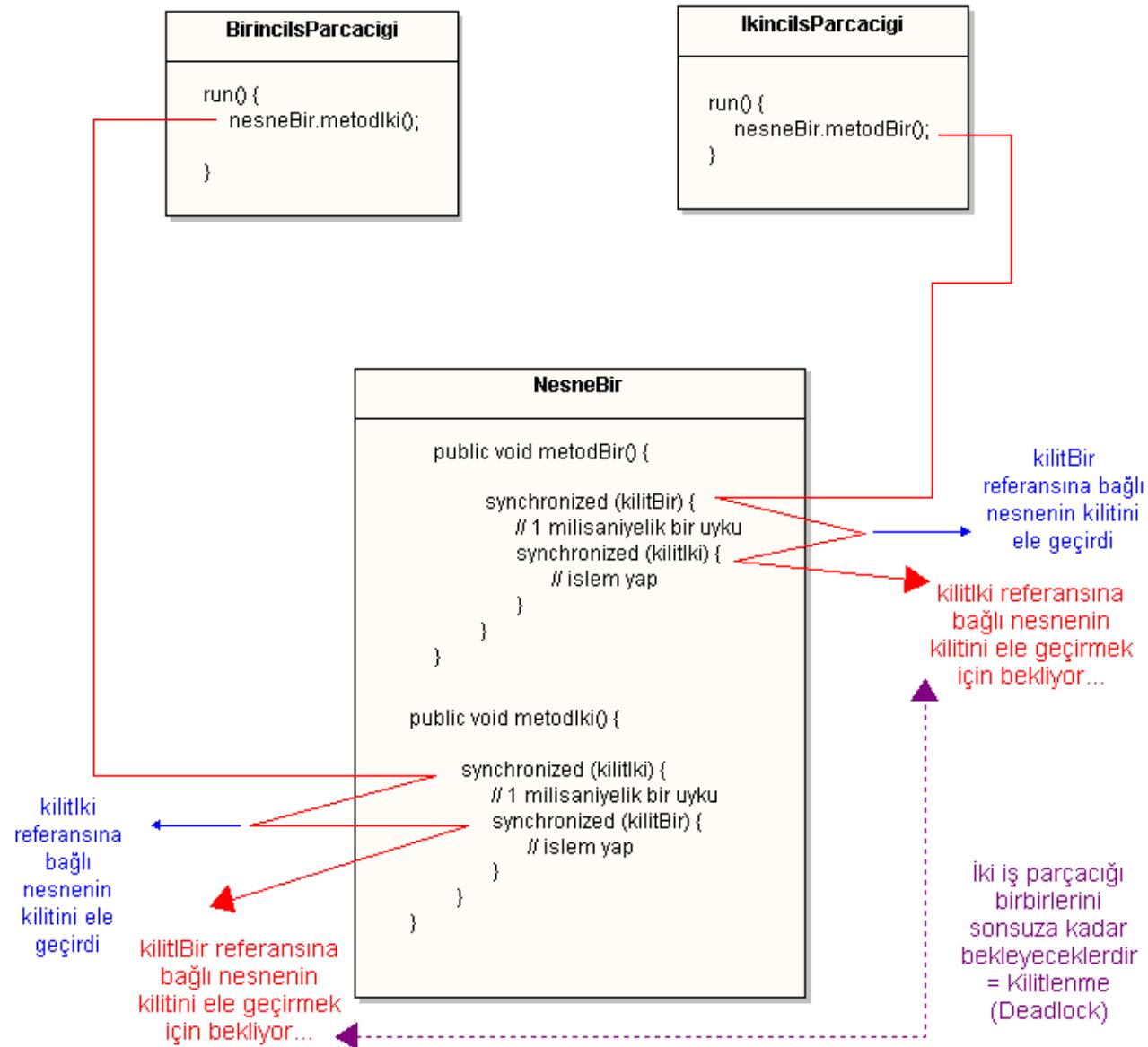
Kilitlenme (Deadlock) - I

- Kilitlenme, ilgili iş parçacıklarının sonsuza kadar beklemesi anlamına gelir.
- Bunun sebeplerinden biri, iki iş parçacığının karşılıklı nesne kilitlerini beklemesinden kaynaklanır.

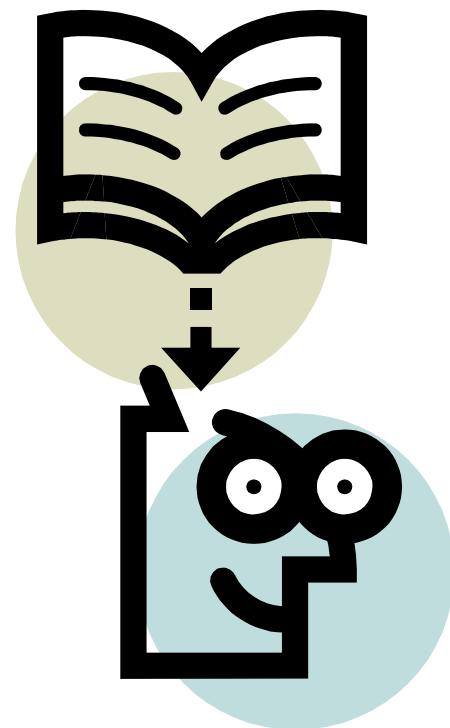
Kilitlenme (Deadlock) - II



Kilitlenme.java

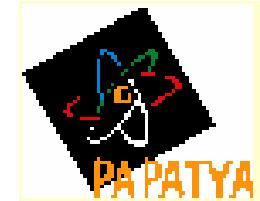
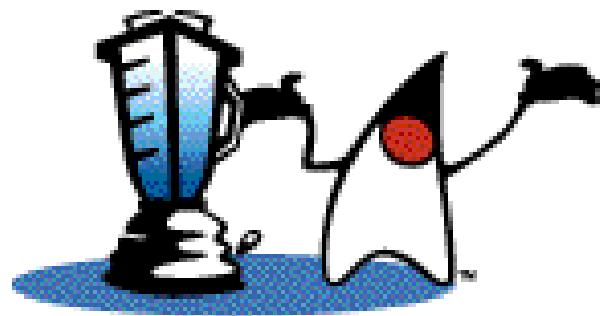


Sorular ...



Nesneler için torbalar

(Collections)



NESNELER İÇİN TORBALAR

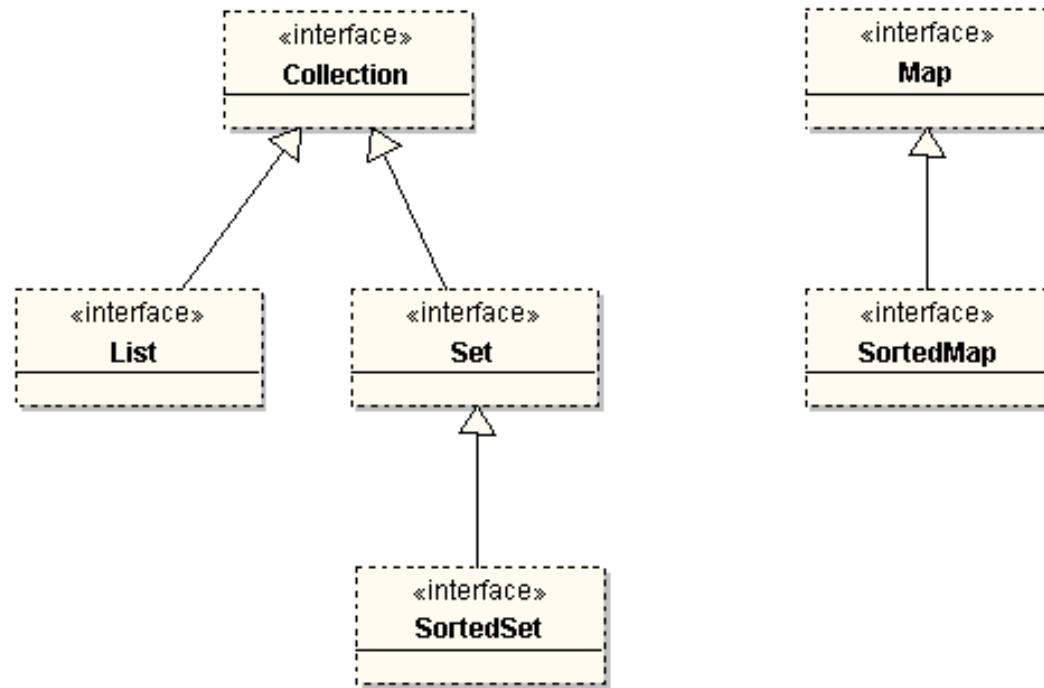
- Torbalar birden çok nesneyi aynı çatı altında toplamak için kullanılır.
- Bunun faydası torba içerisinde bulunan nesnelerin daha kolay taşınmasıdır.
- En basit torba dizilerdir.



DiziOrnekBir.java

Torba Sistemleri

- Bir uygulama yazarken çoğu zaman ne kadarlık bir verinin dizi içeresine konacağı kestirilemez.
- Bu probleme çözüm olarak *java.util* paketinin altındaki arayüzler ve sınıflar kullanılabilir.



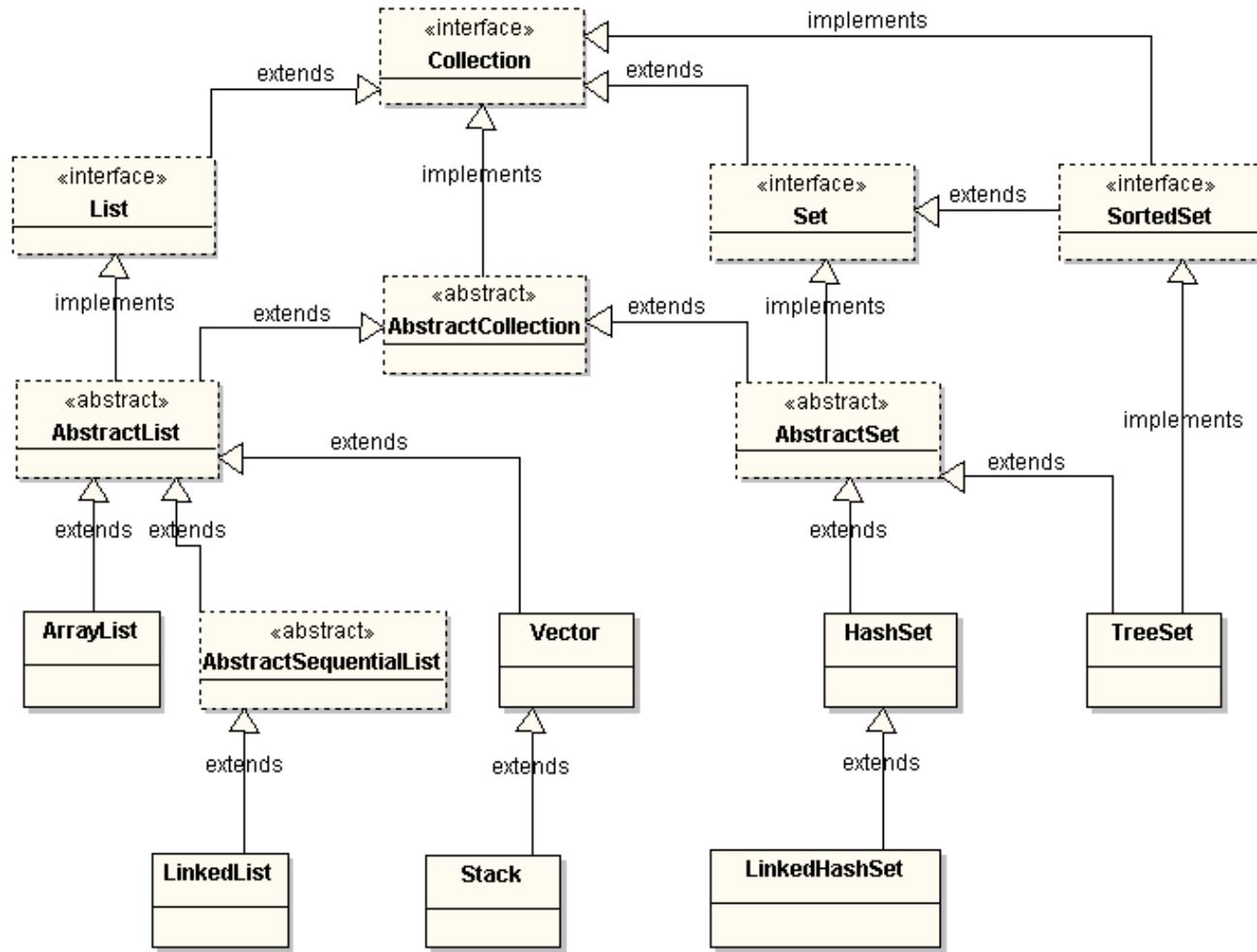
Collection Arayüzü

- *Collection* arayüzüne erişen sınıfların bir kısmı kendisine gelen tüm nesneleri (aynı olsalar da) kabul ederken, kimisi tamamen ayrı nesneler kabul etmektedir.
- Yine bu arayüze erişen bazı sınıflar, içerisindeki elemanları sıralı şekilde tutarken kimisi sırasız bir şekilde tutmaktadır.

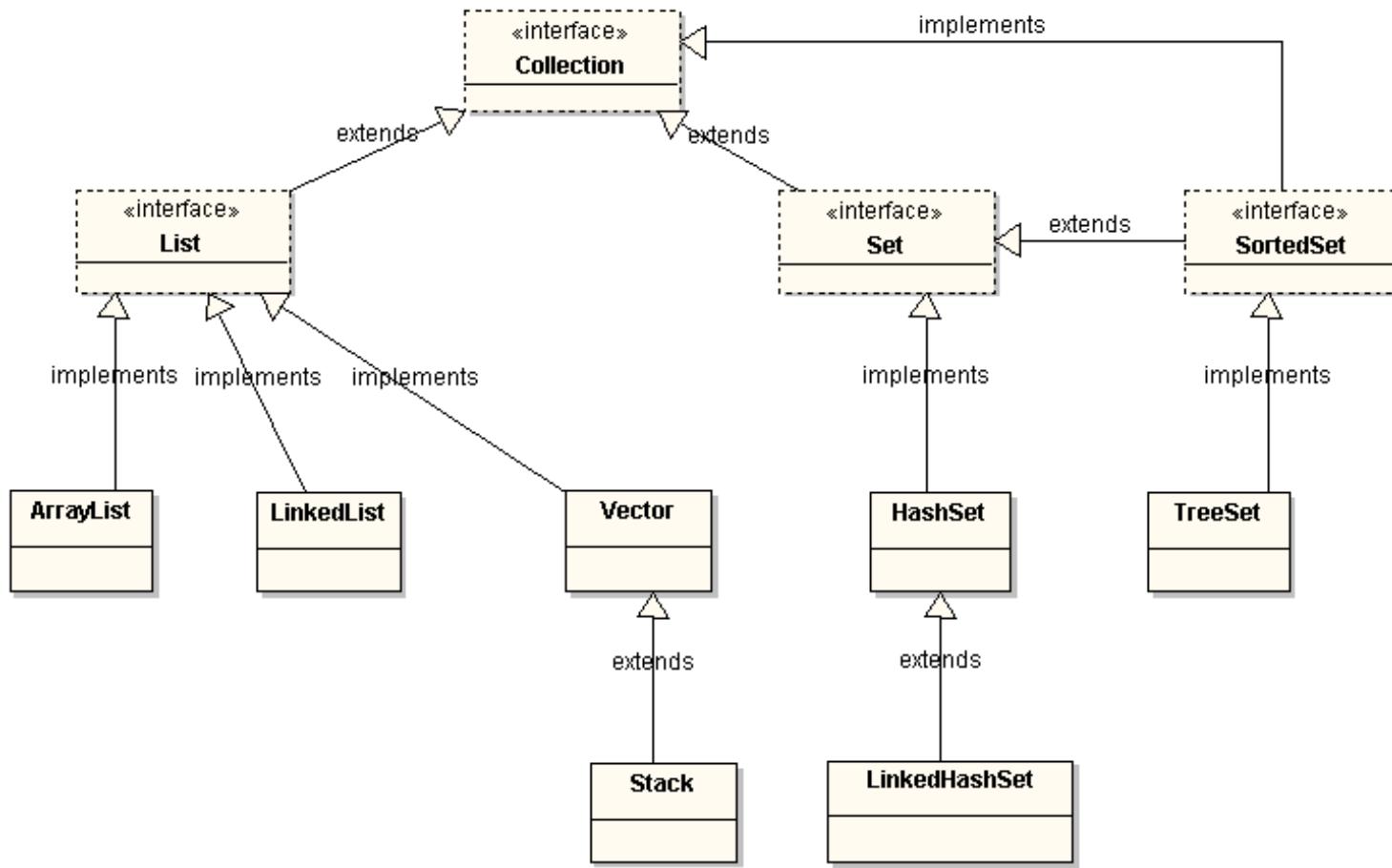


CollectionTest.java

Detaylı Şema



Collection arayüzüne erişen diğer arayüzler ve sınıflar – Detaysız Şema (Soyut sınıflar çıkartılmış)



List Arayüzüne Erişen Sınıflar – ArrayList Sınıfı

- Genel olarak *List* arayüzüne erişen sınıflara ait nesnelerin kullanımı basittir.
- *List* arayüzüne erişen sınıflar, aynı diziler gibi sıfırıncı indeksten başlarlar.
- *ArrayList* nesnesinin içerisinde eleman atmak için **add ()**, içerisindeki bir elemanı almak için ise **get ()** yordamı kullanılır.

```
yeniBoyut= (eskiBoyut*3) / 2 +1
```

ArrayList Sınıfı ve Iterator Arayüzü

- *ArrayList* sınıfı denince akla hemen *Iterator* arayüzüne erişmiş nesneler gelir.
- *Iterator* arayüzü tipindeki nesneler gerçekten çok basit ve kullanışlıdır.

Yordam İsmi	Açıklama
boolean hasNext()	İçeride hala eleman var ise true cevabını geri döner.
Object next()	Bir sonraki elemanı çağırır.
void remove()	next() yordamı ile çağrılmış olan elemanı siler. Bu yordam next() yordamından sonra çağrılmalıdır.



NufusCalismasi.java

Acaba Torbaya Ne Koymuşum?

- *ArrayList* nesnesinin içérisine atılan nesneleri almak için **get()** yordamı kullanılır.
- Bu yordam, içérideki nesneleri *Object* sınıfı tipinde bizlere geri döner.
- Gerçek tipi *Object* sınıfı tipinde olmayan bu nesnelerimizi daha sonradan aşağıya çevirim (*downcasting*) ile gerçek tiplerine çevirmemiz gereklidir.



SuperMarket.java

Garantili Torbalar

```
import java.util.ArrayList;
import java.util.Iterator;

public class StringArrayList {

    private ArrayList al = new ArrayList();

    public void add(String s) {
        al.add(s);
    }

    public String get(int indeks) {
        return (String)al.get(indeks);
    }

    public Iterator iteratorAl() {
        return al.iterator();
    }
}
```

LinkedList Sınıfı

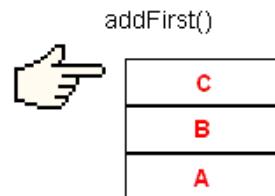
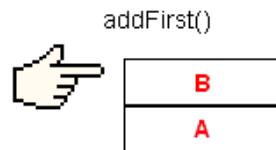
- *List* arayüzüne erişen bir başka sınıf ise *LinkedList* sınıfıdır.
- Bu sınıf da aynı *ArrayList* sınıfı gibi nesnelerin toplu olarak taşınmasında görev alır.
- *LinkedList* sınıfının *ArrayList* sınıfına göre bazı gelişmiş özellikleri bulunur.



LinkedListTestBir.java

LinkedList Sınıfı Kullanarak Yığın Yapısı Oluşturmak

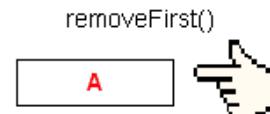
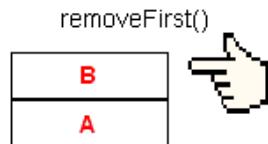
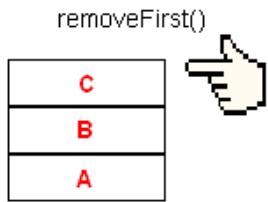
- LinkedList* sınıfı tipindeki nesneye ait olan **addFirst()** ve **removeFirst()** yordamları kullanılarak veri yapılarındaki yığın yapısını tasarlamak mümkündür. (LIFO- Last in first out).



1) A nesnesi içeriye atıldı

2) B nesnesi içeriye atıldı

3) C nesnesi içeriye atıldı



4) C nesnesi içerikden çekildi

5) B nesnesi içerikden çekildi

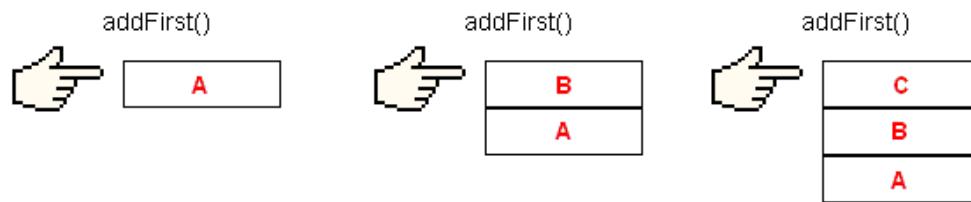
6) A nesnesi içerikden çekildi



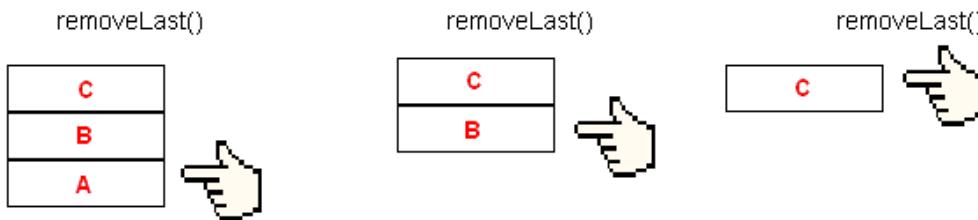
Yigin.java

LinkedList Sınıfı ile Kuyruk Yapısı Oluşturmak

- Kuyruk yapılarındaki kural, içerisinde atılan ilk elemanın yine ilk olarak çıkmasıdır (FIFO-First in first out).



1) A nesnesi içeriye atıldı 2) B nesnesi içeriye atıldı 3) C nesnesi içeriye atıldı



4) A nesnesi içerikden çekildi 5) B nesnesi içerikden çekildi 6) C nesnesi içerikden çekildi



Kuyruk.java

- *Collections* sınıfının *Collection* arayüzü ile kalıtımsal herhangi bir bağlı yoktur.
- *Collections* sınıfının içerisinde bir çok faydalı statik yordam bulunur.
- Bu yordamlar sayesinde *Collection* veya *Map* arayüzüne erişmiş sınıflara ait nesnelerin içerisinde bulunan elemanları sıralama, arama, en büyük elemanı ve en küçük elemanı bulma, v.b. işlemleri gerçekleştirmemiz mümkün olur.

Collections.sort

- Bir torba (ör : *ArrayList* nesnesi) içerisindeki elemanları küçükten büyüğe doğru sıralamak (veya tam ters sırada) için *Collections* sınıfına ait statik **sort()** yordamını kullanabiliriz.



SiralamaBir.java



TerstenSiralama.java



NesneSiralama.java

Soru ?

- Peki *String* veya *Integer* sınıf tipindeki nesnelere referansları *ArrayList* nesnesinin içerisine atmasak da bunun yerine kendi oluşturduğumuz ayrı bir sınıfa ait nesnenin referanslarını *ArrayList* nesnesinin içerisine atsak ve **Collections.sort()** yordamı ile sıralatmaya çalışırsak ne olur?



NesneSiralama.java



Kitap.java



OzgunSiralama.java

java.lang.Comparable

- **compareTo()** yordamının döndürmesi gereken sonuçlar.

Durum	Döndürülen sonuç
O anki nesne (<i>this</i>), parametre olarak gelen nesneden küçükse	O anki nesne (<i>this</i>), parametre olarak gelen nesneden küçükse
O anki nesne (<i>this</i>), parametre olarak gelen nesneye eşitse	sıfır
O anki nesne (<i>this</i>), parametre olarak gelen nesneden büyükse	pozitif tamsayı



Kitap2.java



OzgunSiralama2.java

Collections.min() ve Collections.max()

- Torba (ör:*ArrayList*) içerisindeki elemanların en büyükünü ve en küçüğünü bulan *Collections* sınıfının statik olan **max()** ve **min()** statik yordamlarıdır.



MinMaxBulma.java

Collections.binarySearch()

- *Collections* sınıfı içerisinde bulunan statik **binarySearch()** yordamı ile arama işlemleri kolaylıkla yapılabilir.
- **binarySearch()** yordamı iki adet parametre alır.
 1. Birincisi arama yapılacak olan torba (ör: *ArrayList*)nesnesine ait referans.
 2. İkincisi ise aratılan nesneye ait referans.



AramaTestBir.java

Hangisi Daha Hızlı ArrayList Sınıfı mı, LinkedList Sınıfı mı?

- Şu ana kadar olan örneklerimizin bazlarında *ArrayList* sınıfı bazlarında ise *LinkedList* sınıfı kullanılmıştır.
- Bu iki sınıfın amacı diğer nesnelerin toplanması için torba görevi görmektir.



HızTesti1.java



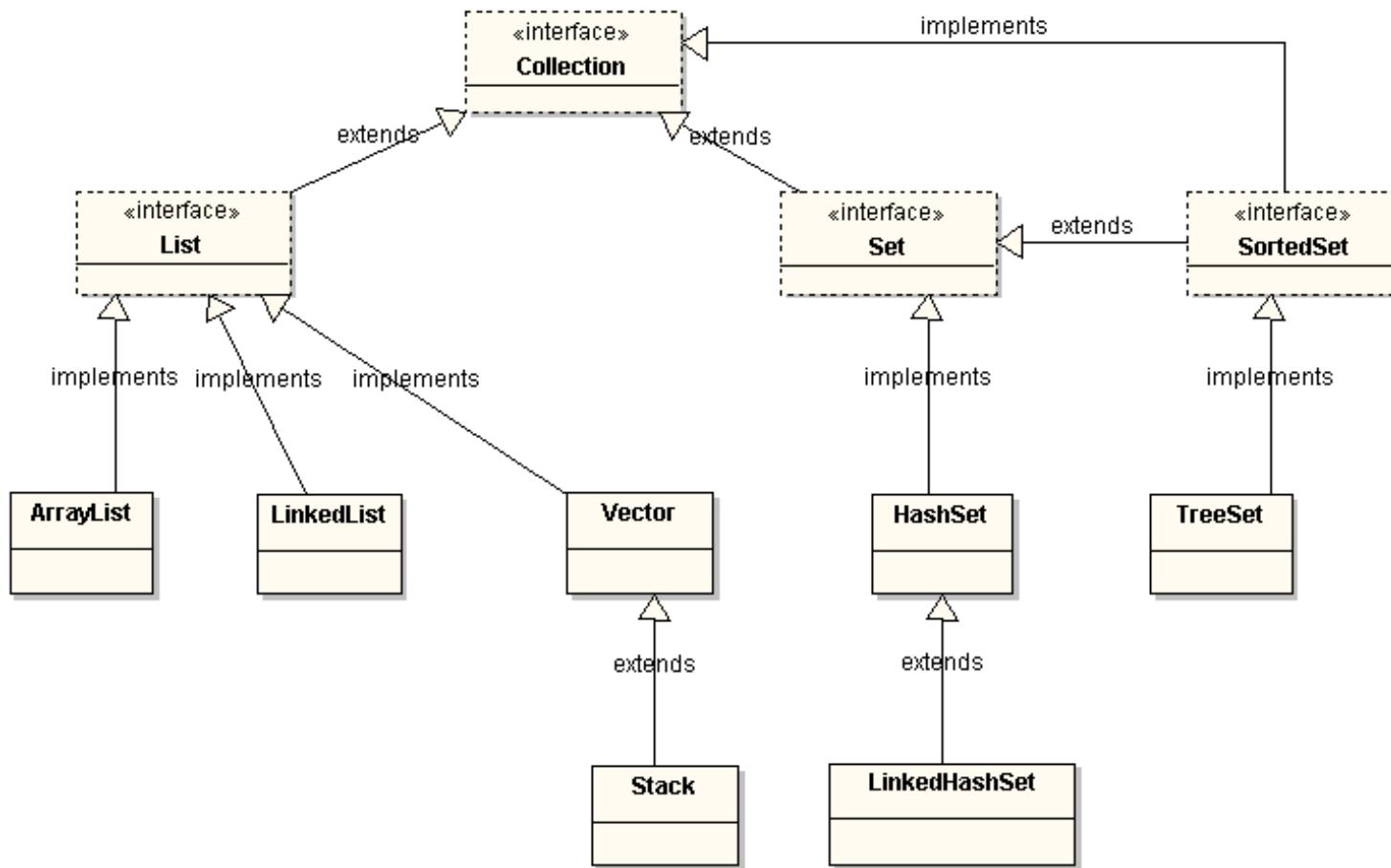
HızTesti2.java

Sonuç

- Arama işlemlerinde *ArrayList* sınıfı en iyi performansı verir.
 - *ArrayList* sınıfı, *RandomAccess* arayüzüne erişir ama *LinkedList* erişmez ve bu yüzden **Collections.binarySearch()** yordamı *ArrayList* üzerinde en iyi performansı verir.
- *LinkedList* sınıfınınında iyi olduğu yerler vardır.
 - Örneğin ters çevirme işlemi -ki bu işlem için **Collections.reverse()** yordamı kullanılır;
 - Ayrıca elemanlar arasında baştan sona veya sondan başa doğru sıralama (*iteration*) işlemlerinde de
 - Eleman ekleme çıkartma işlemlerinde *LinkedList* sınıfı kullanılır

- Set arayüzü *Collection* arayüzünden türetilmiş.
- Set arayüzüne erişen sınıflara ait nesnelerin içerisine aynı elemanı iki kere atamayız.
- Birbirine eşit iki nesneye ait referansı, Set arayüzüne erişen bir nesnenin içerisine atamayız.

Collection arayüzüne erişen diğer arayüzler ve sınıflar – Detaysız Şema



HashSet Sınıfı

- *HashSet* sınıfı *Set* arayüzüne erişmiştir.
- Bunun doğal sonucu olarak da *Set* arayüzü içerisindeki gövdesiz yordamlara gövde yazmıştır.



HashSetTestBir.java

TreeSet Sınıfı

- *TreeSet* sınıfı *SortedSet* arayüzüne erişmiştir.
- *TreeSet* sınıfına ait bir nesnenin özelliği, içerisindeki elemanları sıralı (artan sırada) bir şekilde tutmasıdır.
- *TreeSet* sınıfına ait nesnenin içerisinde atılacak olan referanslara bağlı nesnelere ait sınıfların kesin olarak *Comparator* arayüzüne erişmiş ve bu arayüzün içerisindeki gövdesiz yordamları iptal etmeleri gerekmektedir.



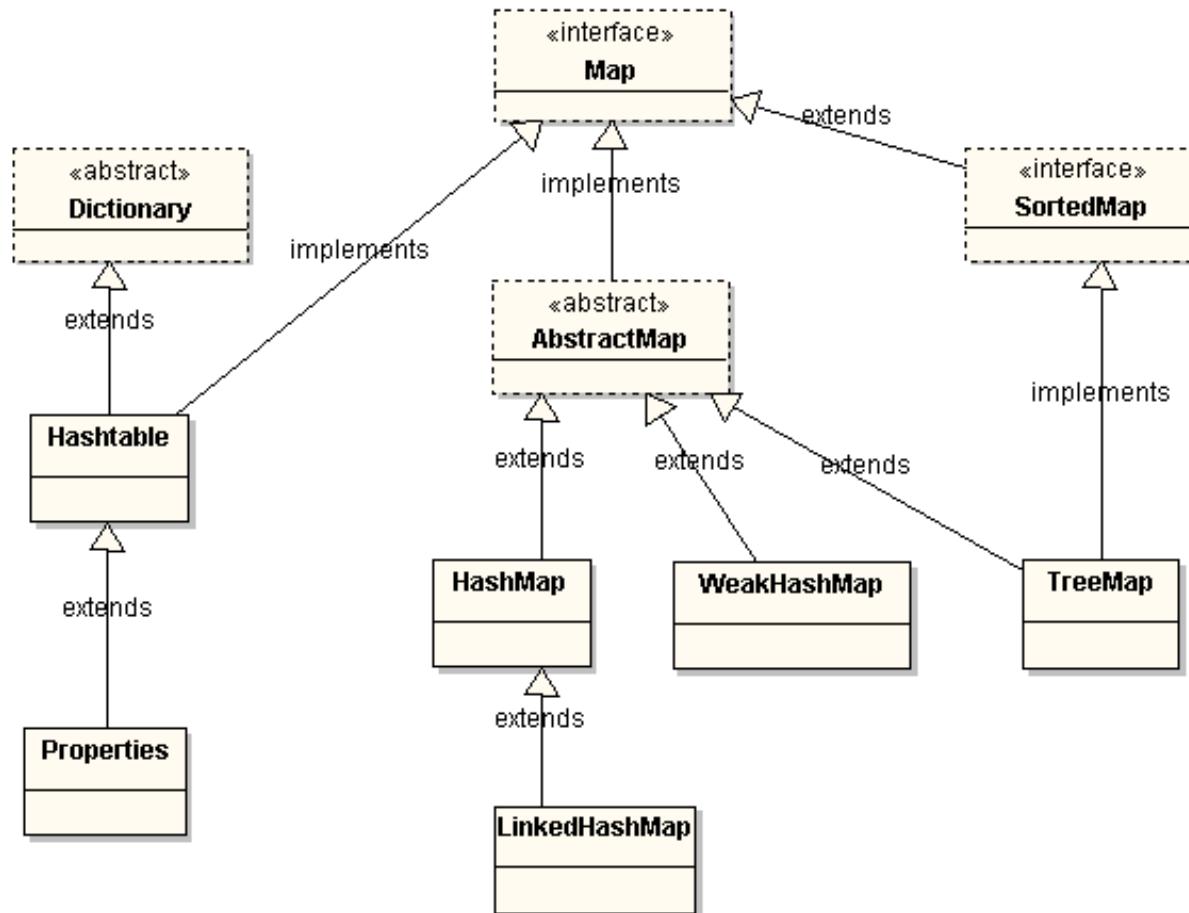
TreeSetTestBir.java

Map Arayüzü

- Uygulama yazarken ihtiyaç duyulan en büyük ihtiyaçlardan biri de anahtar-değer (*key-value*) ilişkisidir.
- Bu anahtar-değer ilişkisini ufak bir veritabanı gibi düşünebilirsiniz.

Plaka kodu (anahtar)	İl (değer)
06	Ankara
07	Antalya
41	Kocaeli
34	İstanbul
77	Yalova
....

Map arayüzüne erişen arayüz, soyut sınıflar ve sınıflar



HashMap Sınıfı

- *HashMap* sınıfı *Map* arayüzüne erişen sınıflarımızdan bir tanesidir.
- Bu sınıfın rolü, kodu yazan kişiye anahtar-değer ilişkisi oluşturabileceği bir ortam sunmaktır.



HashMapTestBir.java



HashMapTestIki.java

HashMap sınıfına ait bir nesneyi bir çok iş için kullanabiliriz.

- Bu tablodan Müşteri1, Müşteri2 ve Müşteri3'ün yaptıkları ödemelerin toplamını kolayca nasıl bulabiliriz ?

No	Müşteri adı	Ödenen tutar
1	Müşteri3	10
2	Müşteri2	5
3	Müşteri2	140
4	Müşteri3	5
5	Müşteri1	90
6	Müşteri2	15
7	Müşteri1	40
8	Müşteri3	65
9	Müşteri2	15
10	Müşteri1	25



Musteri.java



Kasa.java

Hangisi Daha Hızlı ArrayList Sınıfı mı, HashMap Sınıfı mı?

- *ArrayList* ve *HashMap* her ne kadar farklı yapıda olsalar da biri diğerinin yerine kullanılabilir.



HızTesti3.java

- *TreeMap* sınıfı *SortedMap* arayüzüne erişir.
- *TreeMap* sınıfına ait bir nesne kullanılarak aynı *HashMap* sınıfına ait nesnelerde olduğu gibi anahtar-değer ilişkilerini saklamak mümkündür.
- *TreeMap* sınıfına ait bir nesne kullanmanın avantajı anahtar-değer ilişkisindeki anahtarın sıralı bir biçimde tutulmasıdır.



TreeMapTestBir.java

Hangisi Daha Hızlı: HashMap Sınıfı mı, TreeMap Sınıfı mı?

- Bu sorunun cevabı şu şekilde olabilir:
"anahtar-değer ilişkisindeki anahtarın sıralı olmasını istiyorsam *TreeMap*, aksi takdirde *HashMap* sınıfını kullanırım."
- Olaylara bu iki sınıfı ait nesnelerin içerisindeki anahtarın aratılma hızları açısından bakarsak, acaba olaylar nasıl değişir?



HızTesti4.java

Iterator Arayüzü ve Dikkat Edilmesi Gereken Hususlar

- Bu arayüz tipinde bir nesne elde etmek için *Collection* arayüzüne erişen sınıflara ait nesnelerin iterator() yordamını çağrırmak yeterlidir.
- *Iterator* arayüzüne erişen bir nesnenin kullanılmasının sebebi kolaylıktır.
- *Iterator* arayüzüne ait nesneler ilgili torbanın elemanlarının belli bir andaki fotoğrafını çekip daha sonradan bu elemanları başka bir yere kopyalayıp, onların üzerinde mi işlem yapıyor?



IterationOrnek.java

Senkronize Torbalar

- Şu ana kadar incelediğimiz sınıflara ait nesnelerin hiç biri senkronize değildi.
- Bunun anlamı bu nesnelere (*ArrayList*, *LinkedList*, *HashSet*, *HashMap*...) aynı anda iki veya daha fazla sayıdaki iş parçacığının(*threads*) erişip istedikleri eklemeyi veya silme işlemini yapabileceğidir.

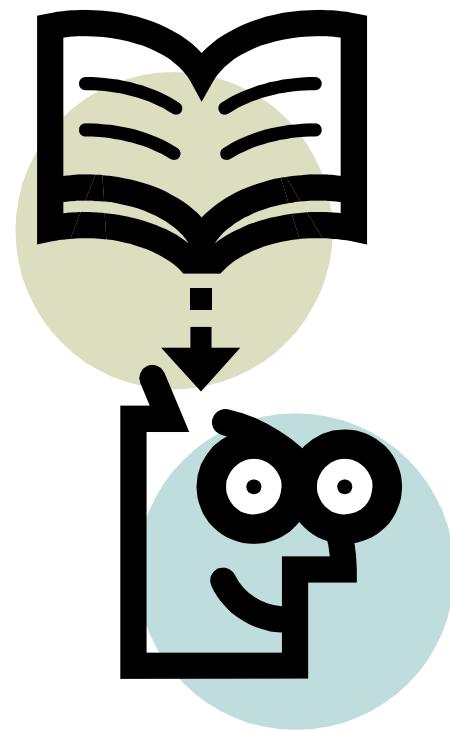


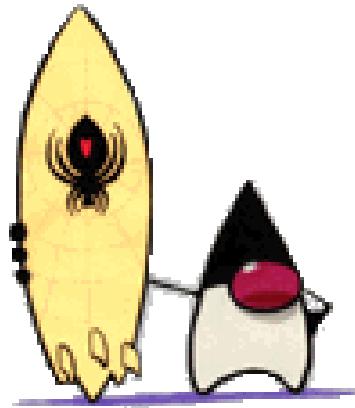
SenkronizeListTestBir.java



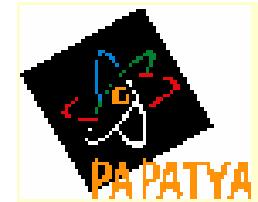
SenkronizeListTestIki.java

Sorular ...





Ağ programlama *(Network programming)*



- Ağ programlama, uygulamaların ağ ortamı üzerinden iletişimde bulunarak veri alış-verişi yapılmasına olanak verir; bu nedenle ağ programlama uygulamalarda önemli bir yer tutar.
- Ağ üzerinde uygulama geliştiren tasarımcı, ağ sisteminin yapısını bilmesi gereklidir; bu yüzden, Java programlama diliyle ağ programlama nasıl yapılır sorusu sorulmadan önce ağ sistemine bir bakmak yararlı olacaktır.

Kullanıcı/Sunucu (*Client/Server*) Nedir? - I

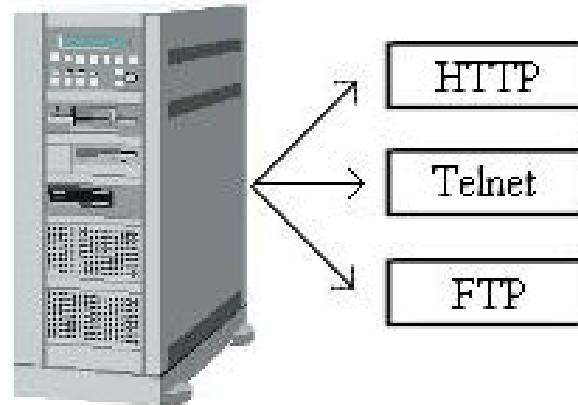
- Kullanıcı, adından da anlaşılacağı gibi bir yerlerden veri almak isteyen uygulamalara/sistemlere verilen isimdir. Örneğin, ağ tarayıcısı en çok bilinen kullanıcı uygulamasıdır.
- Kullanıcı uygulamalarının karşı tarafında bulunan ve aynı bir garson gibi hizmet veren uygulamalasunucu denir.
- Sunucu uygulamasına örnek olarak, yılların efsanevi ağ sunucu uygulaması olan *Apache* verilebilir (<http://www.apache.org>).

Kullanıcı/Sunucu (*Client/Server*) Nedir? - II

- Ağ sunucusu (Web server)
- FTP (File Transfer Protocol) sunucusu
- Telnet sunucusu



İstemci (Client)



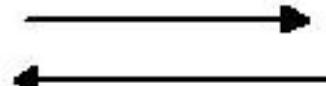
Sunucu (Server)

IP Adresi ve Port Kavramları - I

- Internet'e bağlı her sistemin kendisine ait özel bir numarası vardır -ki bu numaraya IP adresi denilmektedir.
- Örneğin <http://www.kodcu.com> sitesini barındıran sunucu sistemin IP adresi 212.115.21.14'tür.
- İnsan belleği sayısal IP adresleri anımsamakta zorluk çeker; dolayısıyla, ayrıca alan adı sistemi mevcuttur (*Domain Name System*). Alan adlarına örnek olarak www.kodcu.com veya www.riskturk.com isimleri verilebilir.
- Bir IP adresine birden çok alan adı bağlanabilmektedir.

IP Adresi ve Port Kavramları - II

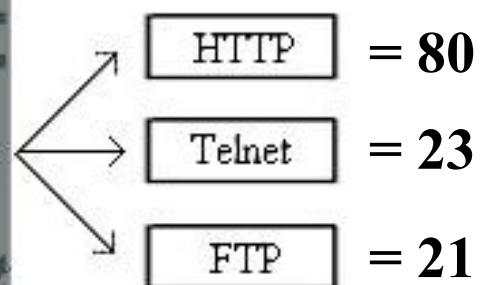
- Birçok sunucu uygulamalarının aynı sistem üzerinde çalışmaları, onların aynı IP adresinden hizmet verdiklerini gösterir; ancak, bu sunucu uygulamalarının birbirinden farkı herbirimin farklı port numaralarından hizmet vermeleridir...



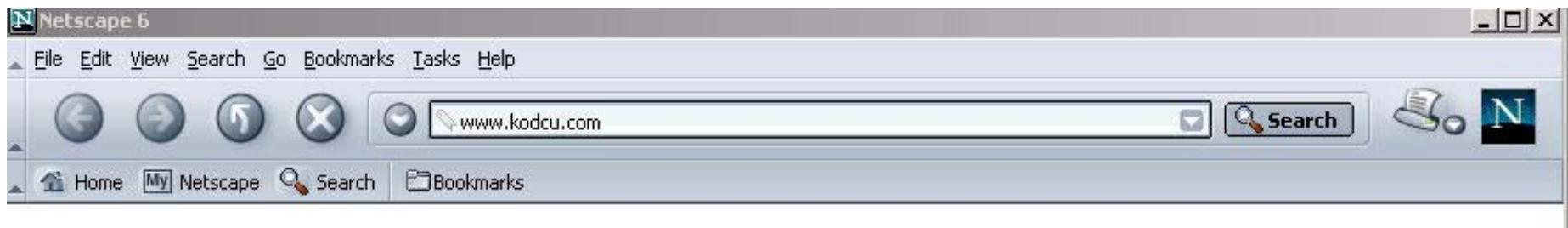
İstemci (Client)



Sunucu (Server)



Ağ sunucusu 8080. port numarasından hizmet verebilir mi ?



Soket Nedir?

- Soket, IP adresi ve port numarasının birleşiminden oluşan ve uygulama yazılırken ağ içerisindeki altdüzeydeki ayrıntılardan kurtulup, iki sistemin birbirine kolayca bağlanmasını sağlayan bir tanımlama/kavramdır.



- Ağ üzerinde çalışan uygulama geliştirilebilmesi için geliştirme aşamasında belirli bir IP adresine sahip sistem üzerinde çalıştırılması gereklidir.
- Peki ağ üzerinde çalışacak bir uygulama yazılması için her seferinde Internet'e/ağa bağlı olunması mı gerekiyor?



IPTest.java

IPv4 ve IPv6 Uyarlamaları - I

- Şimdilerde ortada IP adres yetmezliği gündeme gelmiştir.
- Eski sistem yani aslında şu an için kullandığımız IP adres sistemine IPv4 denilmektedir.
- Bu sistemde IP adresleri ondalık 32-bit işaretetsiz sayılarından oluşmaktadır, IPv4 sisteminin iskeleti **aaa.bbb.ccc.ddd** şeklinde bulunmaktadır.
- Biraz önce incelenen örnekte bahsi geçen IP adresi IPv4 sistemindeydi, 195.155.246.159 gibi...

- IPv6 sistemi yönlendirme (*routing*), otomatik adres konfigürasyonu, doğrulama (*authentication*), gizlilik (*privacy*) ve taşınır IP (IP mobility) gibi bir çok yeniliği de beraberinde getirmektedir.
- IPv6 sistemi ise onaltılık (*hexadecimal*) 128-bit işaretetsiz sayılardan oluşmaktadır.

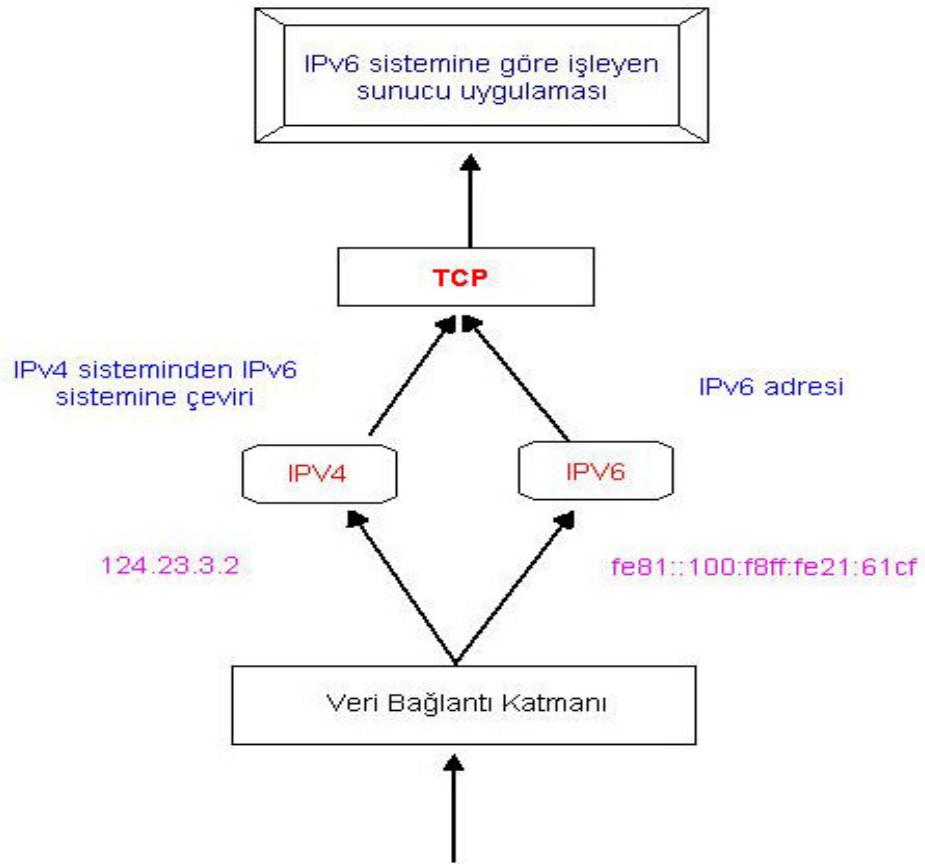
Ör:

```
3ffe:1700:4242:7:500:f8cf:fe21:67cf
```

- Java Programlama dilinde IP adresini ifade etmek için *InetAddress* sınıfı kullanılır.
- Bu sınıf J2SE 1.4’te (*Java 2 Standart Edition*) bir çok değişikliğe uğramıştır.
- İşin aslı, Java programlama dilini kullanarak uygulama yazan bir kişi için IP adresinin IPv4 veya IPv6 sistemine göre olmasının fazla bir önemi yoktur çünkü *InetAddress* sınıfı arka planda tüm işleri bizim yerimize halleter.

IPv4 veya IPv6 sistemleri - I

```
InetAddress ipAdresi = InetAddress.getByName("www.obje7.com");  
Socket s = new Socket(ipAdresi, 80);
```



IPv4 veya IPv6 sistemleri - II

- Bu ifademiz ise bir sunucu uygulamaya aittir. Bu uygulamamız 8080. port numarasından devamlı olarak dinleme yapıp gelen istekleri karşılamak için tasarlanmıştır.

```
ServerSocket sunucu = new ServerSocket(8080);
for (;;) {
    Socket istemci = sunucu.accept;
    // kimin baglandigini ogrenelim
    .....

    InetAddress istemcininAdresi = istemci.getInetAddress;
}
```

- Java programlama dilini kullanarak yapacağımız ilk ağ uygulamalar, basit bir sunucu ve istemci üzerine olacaktır.



SunucuOrnekBir.java

Sıra	Uygulamanın adı	Dinlediği port numarası	Çalışması	Açıklama	Original Hata (Eğer oluşmuş ise)
1.	SunucuOrnekBir.java	8080	java SunucuOrnekBir	Sorun yok, 8080. port numarasından dinlemeye başlar.	
2.	SunucuOrnekBir.java	8080	java SunucuOrnekBir	Sorun var, çünkü 8080. port numarası dolu.	Exception in thread "main" java.net.BindException: Address already in use: JVM_Bind
3.	Tomcat Servlet/JSP container	8080	startup.sh (Linux/Unix için), startup.bat (windows için)	Sorun var, çünkü 8080. port numarası dolu.	Exception in thread "main" java.net.BindException: Address already in use: JVM_Bind

Sunucu/İstemci Uygulamalar – II

- Java programla dilini kullanarak basit bir istemci uygulaması yazmak için sadece java.net paketinin altındaki *Socket* sınıfını kullanmak yeterli olacaktır.



IstemciOrnekBir.java

Detaylar

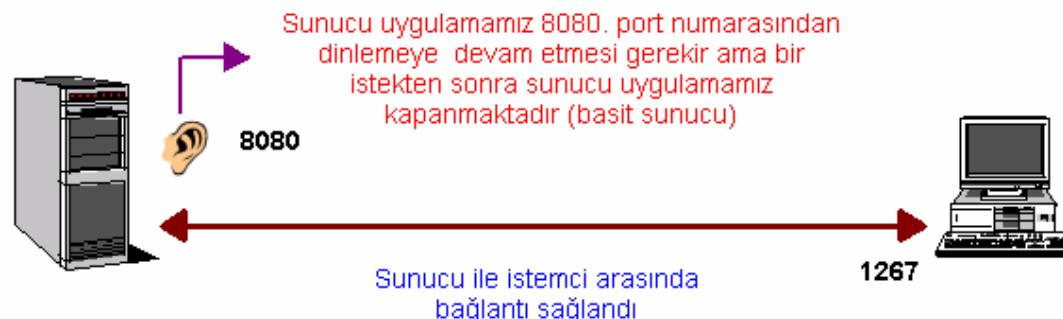
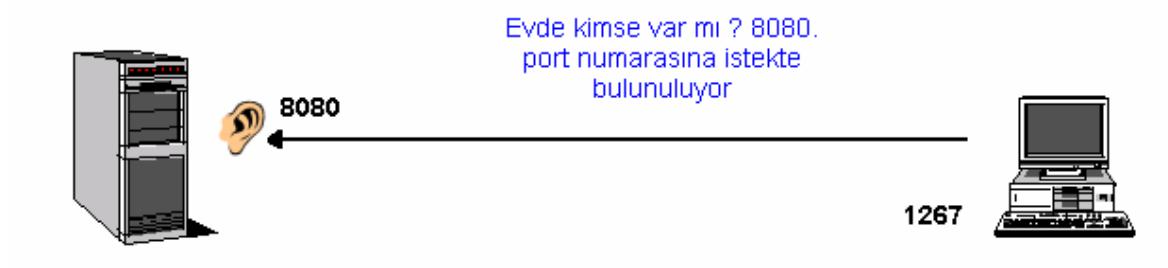
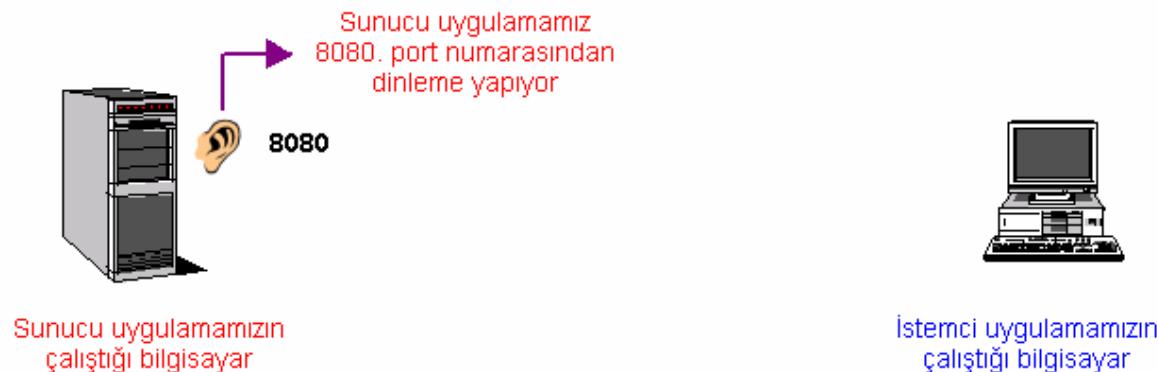
baglanti kabul edildi: Socket[addr=/127.0.0.1,port=1267,localport=8080]

soket = Socket[addr=localhost/127.0.0.1,port=8080,localport=1267]

Aşağıdaki dörtlünün bir tekillik (unique) oluşturulması gereklidir.

- İstemcinin IP adresi
- İstemcinin port numarası
- Sunucunun IP adresi
- Sunucunun port numarası

SunucuOrnekBir.java ve İstemciOrnekBir.java uygulamaları arasındaki ilişki



Bloke Bağlantılar - I

- Bloke bağlantılar, G/Ç (Girdi/Çıktı-I/O-*Input/Output*) işlemi gerçekleştiği anda uygulamayı havada asılı bırakan bağlantılardır; bir başka deyişle uygulamayı bekleten bağlantılardır.
- *SunucuOrnekBir.java* ve *IstemciOrnekBir.java* arasındaki ilişkinin yönlendiricisi istemci uygulamayı çünkü istemci uygulama verileri 300ms yerine 1 saatte gönderirse sunucu uygulama böyle bir hareketin karşısında boyun eğmek zorundadır.



SunucuOrnekIki.java

Bloke Bağlantılar - II

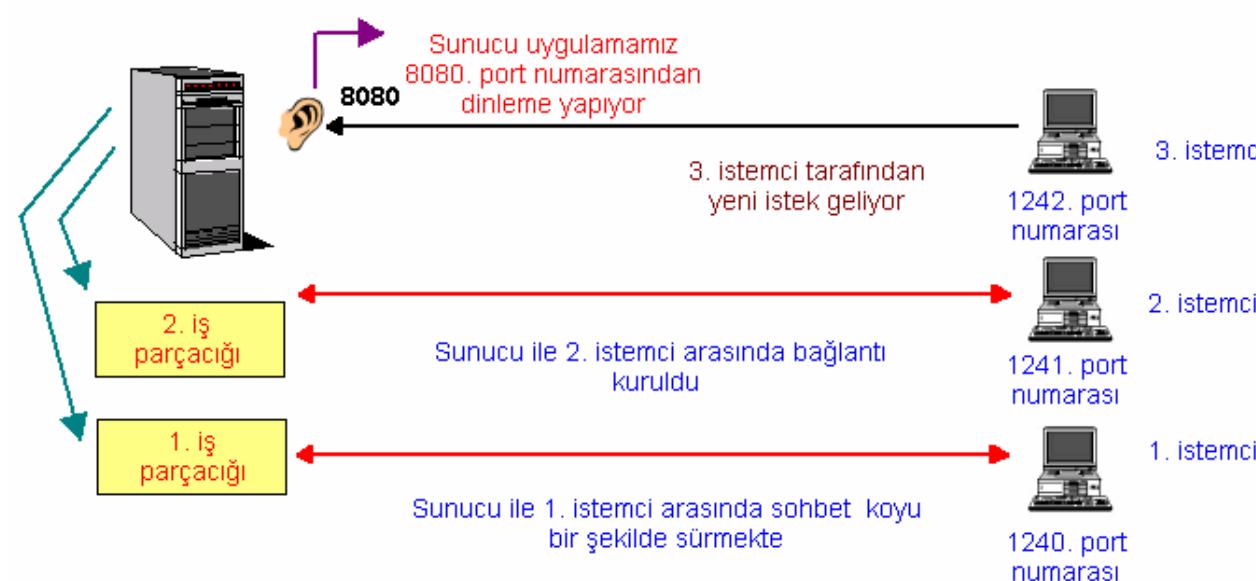
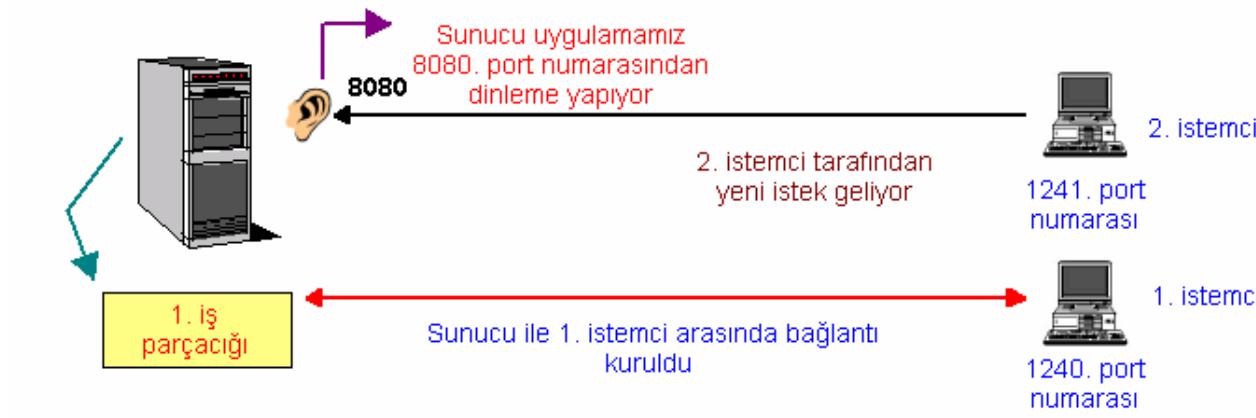
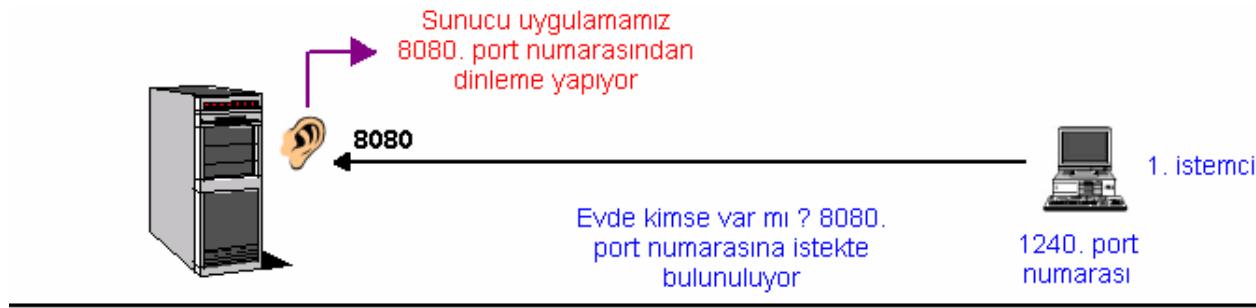
- *SunucuOrnekIki.java* uygulamamızın eksikliği ise kendisine gelen tüm istemcilere teker teker hizmet vermesidir.
- Örneğin iki istemci uygulama bu sunucuya bağlanmak isterlerse, bu istemci uygulamalarından önce biri, daha sonra ise öteki kabul edilecektir.
- Şimdi *IstemciOrnekBir.java* içerisindeki **Thread.sleep** yordamına 300 yerine 10000 yazıp bu istemci uygulamamızı baştan derleyip çalıştırırsak, istemci uygulamalarımız verileri 10000ms bekleterek sunucuya gönderecektir.

İş Parçacıkları (*threads*) İş Başında

- Sunucu uygulamalarının aynı anda birçok isteğe cevap verebilmesi için işin içerisinde iş parçacıklarının da dahil edilmesi gereklidir.

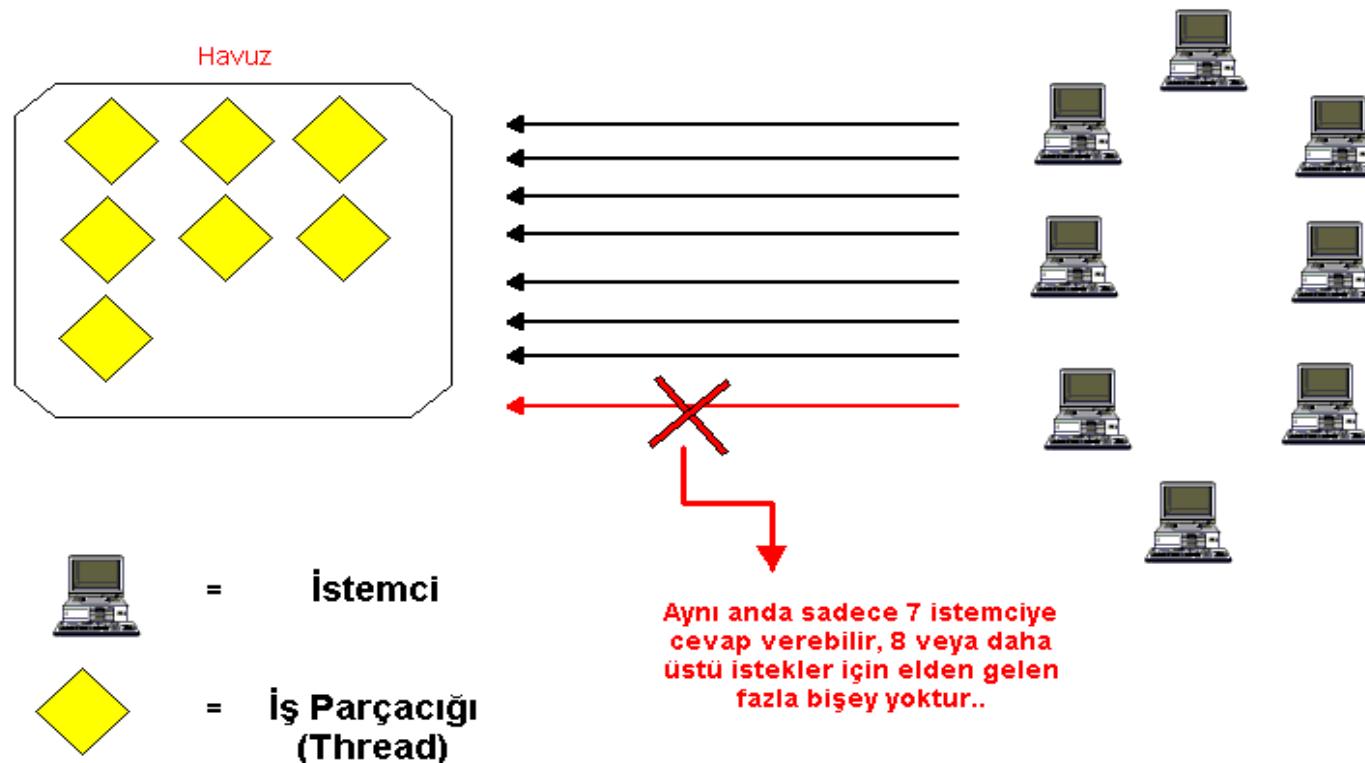


SunucuOrnekUc.java



Havuz

- Her istemci isteği için ayrı bir iş parçacığının oluşturulması çok lükse kaçar.



IstemciOrnekIki.java



SunucuOrnekDort.java

URL ve URLConnection Sınıfları

- URL sınıfının yaptığı başlıca görevlerden biri aynı tarayıcı(browser) gibi çalışmasıdır.

1. mini kurs notlarını bu adresten okuyabilirsiniz :

<http://www.firmanınadresi/minikurs/ders1250.html>

2. mini kurs notlarını bu adresden okuyabilirsiniz :

<http://www.firmanınadresi/minikurs/ders559.html>



SayfaBulucu.java

Veritabanı sunucularına bağlantı

- Veritabanı sunucularına hangi organizasyonların ihtiyacı olabilir ?
 - **Bankalar**
 - **Hastaneler**
 - **Üniversiteler**
 - **Devlet Daireleri**
- Bankalar, müşteri bilgileri
- Hastaneler, hasta kayıtlarını
- Üniversiteler, öğrenci bilgilerini
- Devlet Daireleri, planlama bilgileri, vatandaş bilgileri..gibi

SQL

- SQL (Standart Query Language- Standart Sorgulama Dili), veritabanlarını oluşturmak, kontrol etmek ve yönetmek için kullanılan ve 1992 yılında standart hale getirilmiş olan bir dildir.
 - Veritabanı, kalıcı verileri tablo şeklinde saklayan sistem
 - Tablo, satırlardan oluşan yapı
 - Satır, kolonlardan oluşan yapı
 - Kolon, isim, tip ve değere sahip olan tek bir veri yapısı.

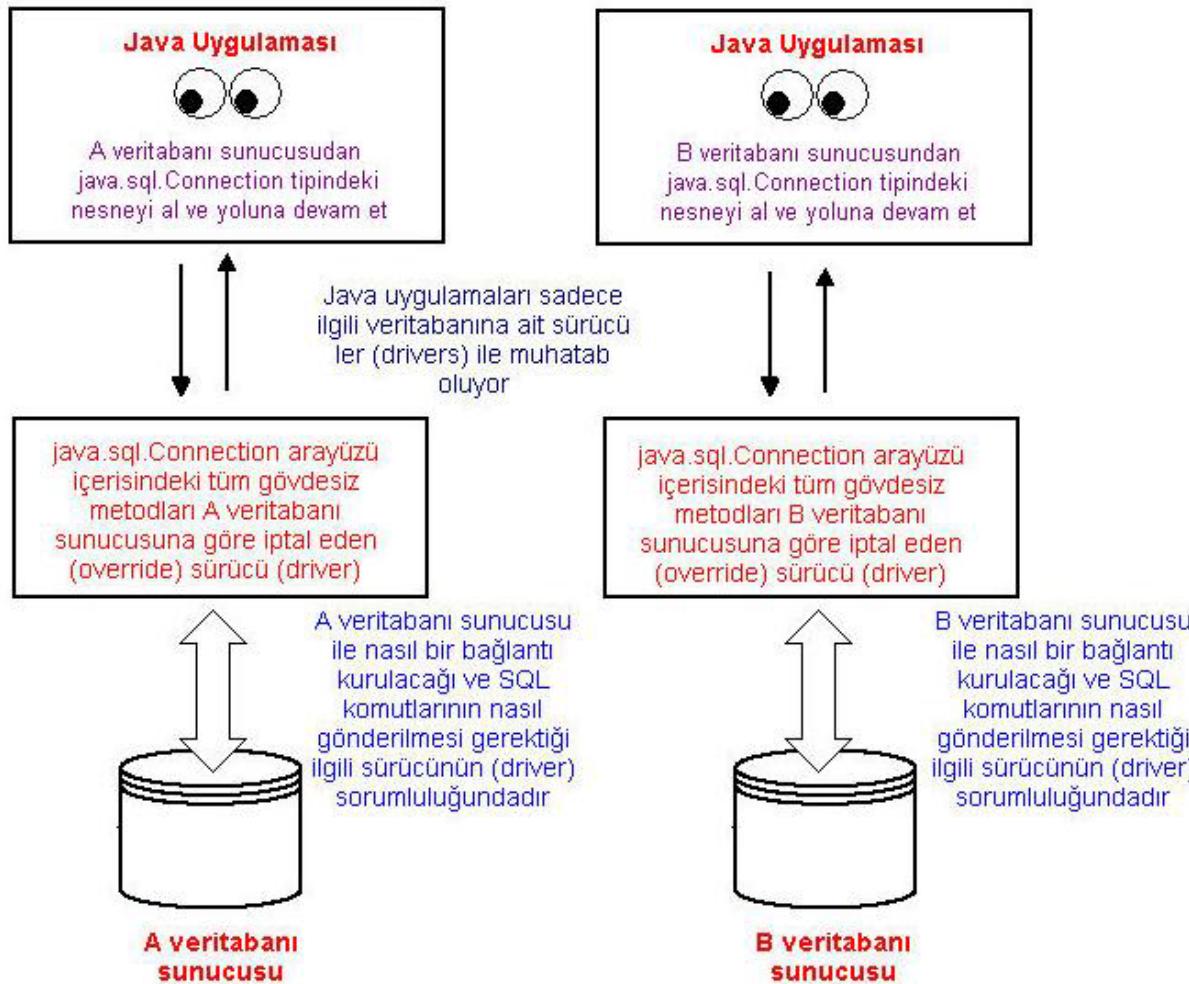
AD	SOYAD	ADRES	TELEFON	CEPTELEFON	EPOSTA	SEHIR	SEMT
OYTUN	IRMAK	ZAMBAK SOK, ERIŞ	0 216 55555678		OYTUN@OYTUNC.COM	ISTANBUL	ERENKOY
ESIN	ALAN	BULVAR CAD. KOK	0 262 55501457	7770700	ESIN@ALAN.COM	IZMIT	MERKEZ
MAHIR	EKİNCİ	CİFTÇIOĞLU SOK. K	0 212 55501132	9990901	MAHIR@MAHIR.CO	ISTANBUL	MECİDİYEKOY
HAKAN	CAMCI	ZİVER SOK, A APT.	0 216 55501148		HAKAN@CAMCI.CO	ISTANBUL	ERENKOY
LEVENT	PAK	İNÖNU CAD, KAS A	0 216 55501151	8880802	PAK@PAK.COM	ISTANBUL	BOSTANCI
EMRE	CAN	BUYUKDERE CAD.	0212 55501169		CAN@CAN.COM	ISTANBUL	MASLAK

MUSTERİ tablosu

JDBC (Java DataBase Connectivity-Java Veritabanı Bağlantılılığı)

- Her veritabanı sunucusunun belli bir üreticisi vardır.
- Şu an için kullanılan en yaygın veritabanı sunucularından bir kaçının aşağıdaki gibidir.
 1. **MySQL** : 3306 port numarasından hizmet verir.
 2. **Oracle** : 1521 port numarasından hizmet verir.
 3. **MSSQL** : 1433 port numarasından hizmet verir.

Genel Bakış



MySQL için gerekli olanlar...

- MySQL açık kaynaklı bir projedir. Bu veritabanı sunucusu GPL (GNU General Public License) lisansı altında bedava kullanılabileceği gibi, belirli bir ücret karşılığında ticari lisans altında da kullanılabilir.
- Sadece MySQL değil tüm veritabanı sunucularına ait sürücülerini (driver) bulabilmek için
<http://www.java.sun.com/products/jdbc/> adresi ziyaret edilebilir.

CLASSPATH ayarları

- Java uygulamalarının MySQL veritabanına sunucusuna bağlanabilmesi için sistemimizde iki şeyin olması gereklidir.
 - MySQL veritabanı
 - MySQL sürücüsü

Linux

```
$ setenv CLASSPATH /kurulum/dizini/mm.mysql.jdbc-[versiyon]:$CLASSPATH
```

Windows

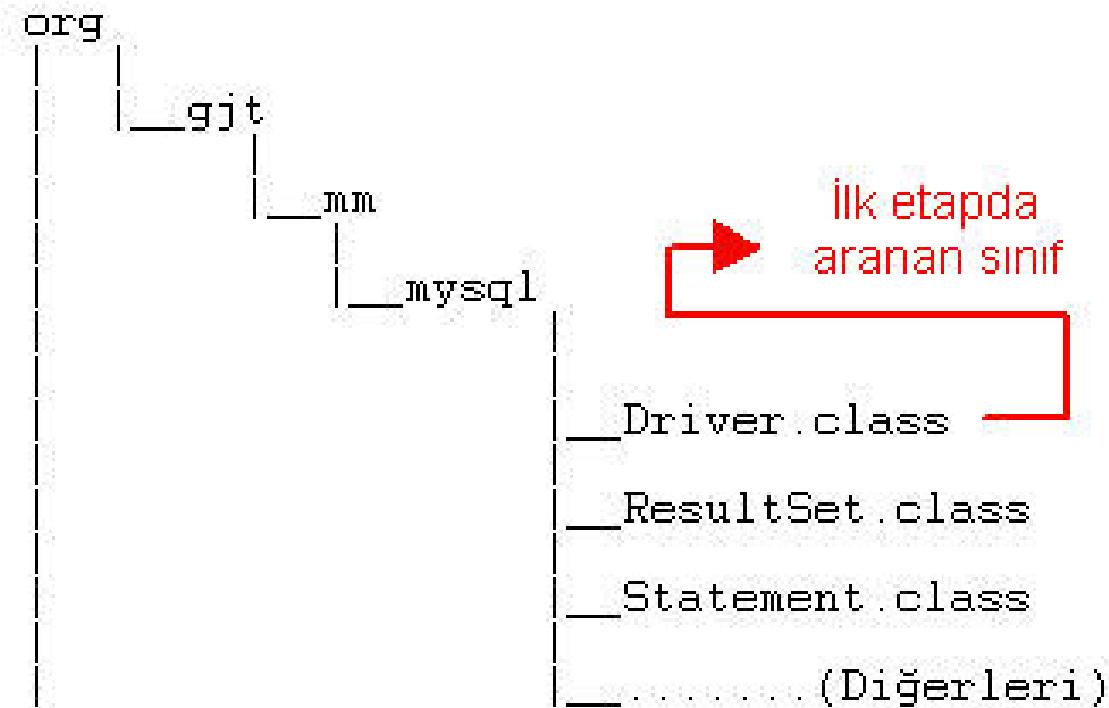
```
C:> set CLASSPATH=\kurulum\dizini\mm.mysql.jdbc-[versiyon];%CLASSPATH%
```



VeritabaniBaglantiTestBir.java

Sürücünün(*Driver*) yapısı

- > **jar -xvf mm.mysql-2.0.10-bin.jar**



Bağlantı

AD	SOYAD	ADRES	TELEFON	CEPTELEFON	EPOSTA	SEHIR	SEMT
OYTUN	IRMAK	ZAMBAK SOK, ERIS	0 216 55555678		OYTUN@OYTUN.COM	ISTANBUL	ERENKOY
ESIN	ALAN	BULVAR CAD, KOK	0 262 55501457	7770700	ESIN@ALAN.COM	IZMIT	MERKEZ
MAHIR	EKİNCİ	CİFTÇIOĞLU SOK, F	0 212 55501132	9990901	MAHIR@MAHIR.COM	ISTANBUL	MECİDİYEKOY
HAKAN	CAMCI	ZİVER SOK, A APT.	0 216 55501148		HAKAN@CAMCI.COM	ISTANBUL	ERENKOY
LEVENT	PAK	İNÖNÜ CAD, KAS A	0 216 55501151	8880802	PAK@PAK.COM	ISTANBUL	BOSTANCI
EMRE	CAN	BUYUKDERE CAD,	0212 55501169		CAN@CAN.COM	ISTANBUL	MASLAK

MUSTERİ tablosu

```
Connection conn = DriverManager.getConnection(  
    "jdbc:mysql://127.0.0.1/test?user=root&password=root");
```

Veritabanı
sunucusunun IP
adresi

Veritabanının
ismi

Kullanıcı isminin belirtildiği
kısım -ki bu örneğimizde
kullanıcı adı = root

Şifrenin belirtildiği kısım
-ki bu örneğimizde
şifre=root



VeritabaniBaglantiTestIki.java

MUSTERI tablosuna yeni bir kayıtın eklenmesi

- **executeUpdate ()** ve **executeQuery ()** yordamlarının arasındaki farklar nelerdir ?



VeritabaniBaglantiTestUc.java

```
#> java VeritabaniBaglantiTestUc ONUR AY "BUYU  
K SITE B BLOK KAT 5 NO 18" "0 262 11112222" "6606645"  
"ONUR@ONUR.COM" "IZMIT" "  
KURUCESME"
```

MUSTERI tablosundaki mevcut kayıtların değiştirilmesi ve silinmesi

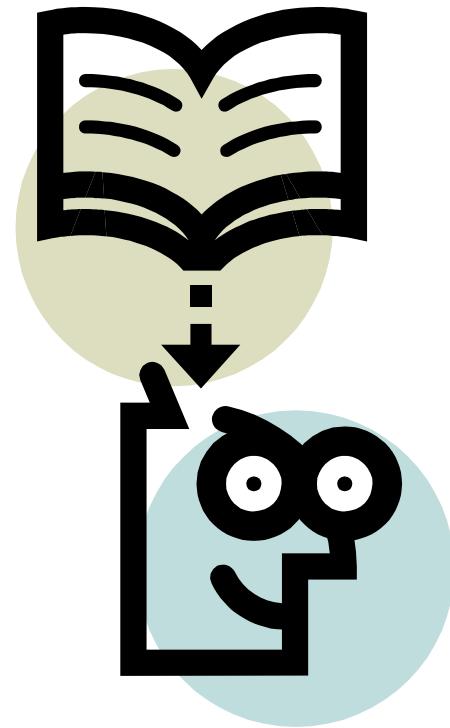


VeritabaniBaglantiTestDort.java



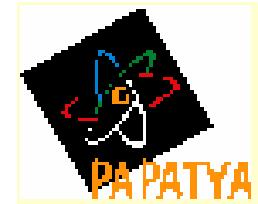
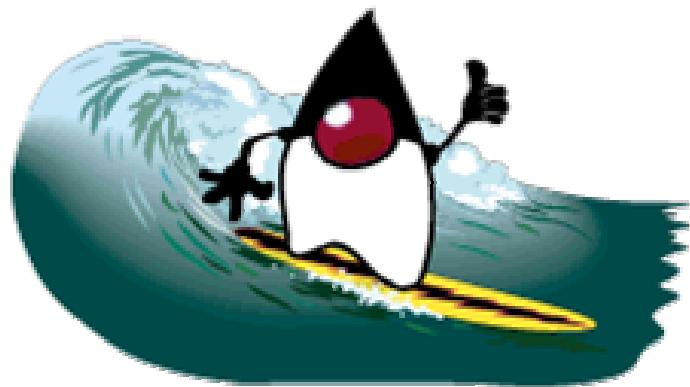
VeritabaniBaglantiTestBes.java

Sorular ...



Hata Ayıklamanın Ötesi...

(Assertion)



Assertion

- Assertion kelimesinin Türkçe karşılığı iddia, birşeylerin doğruluğunu ispat etmek anımlarına gelir.
- Assertion özelliği, J2SE 1.4 versiyonu ile birlikte gelen yeni bir özelliktir.
- Bu yeni gelen özellik sayesinde hata ayıklama (*debugging*) ve yazılan kodların doğruluğunu ispat etme süreçleri çok daha basite indirgenmektedir.

Hata Ayıklama (*Dubugging*) - I

- Hata ayıklamak (*debugging*) ne demek ?
- Hata ayıklama işlemi, hatanın algılanmasından sonra gelen bir süreçtir ve süreci uygulamak için bir çok yöntem bulunur.
- En bilindik yöntemlerden biri hatalı olduğuna inanılan kod yığınlarının arasına **System.out.println()** komutları serpiştirilerek uygulamanın akışı takip edilmeye çalışılır.

Hata Ayıklama (Debugging) - II

```
public double ortalamaBul(double[] dizi) throws Exception{  
    → System.out.println("ortalamaBul metodunun icinde.."); //dikkat  
    if (dizi == null) {  
        → System.out.println("dizi null gelmis, hata fırlatilacak"); //dikkat  
        throw new Exception("Gonderilen dizi null");  
    } else if ( dizi.length == 0) {  
        → System.out.println("dizi icerisinde eleman yok"); //dikkat  
        return 0;  
    }  
  
    double ortalama = 0;  
    for (int i=0; i<dizi.length; i++){  
        → System.out.println("dizi["+i+"] : " + dizi[i]); //dikkat  
        ortalama += dizi[i];  
    }  
    → System.out.println("1- ortalama : " + ortalama); //dikkat  
    ortalama = ortalama / dizi.length;  
    → System.out.println("2- ortalama : " + ortalama); //dikkat  
    return ortalama;  
}
```

Hata Ayıklama (Debugging) - III

- Diğer yöntem ise "*Java Platform Debugger Architecture*" mimarisini kendi içerisinde entegre etmiş bir editör ile çalışmaktadır.
 - Eclipse
 - VisualSlickEdit
 - JBuilder
 - CodeGuide
 - gibi...

Assertion özelliğini kullanmak

- Assertion özelliğini yazılan kodların içerisinde yerleştirmek çok kolaydır.
- Assertion, koşullar gerçekleşmediği zaman hata fırlatan bir mekanizmadır.
- Assertion özelliğini kullanmanın iki yolu vardır.

Assertion özelliğini kullanmak – Birinci Yol

- Birinci yol sadece basit bir ifadeden oluşur.

```
assert ifade ;
```

Yukarıda belirtilen ifade **true** ise sorun çıkmaz ama şayet bu ifade **false** ise sorun var demektir ve hata (AssertionError) fırlatılır.

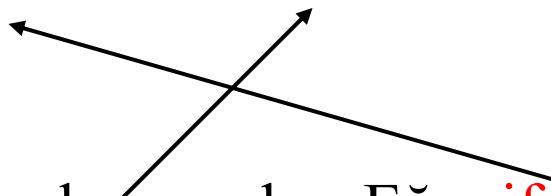
Örnek - Birinci Yol

```
public class OrnekSinif {  
    public void ornekMetod( int parametre ) {  
        Asinifi a = null;  
        // ... Asinifi tipinde bir nesne alınmaya çalışılıyor  
        // İşlem başarılı mı başarısız mı olmuş ?  
        assert a != null; ←—————★  
    }  
}
```

Yukarıdaki assert ifadesinin kullanılmasındaki amaç, Azinifi sınıfı tipindeki **a** referasının acaba Azinifi sınıfına ait bir nesneye mi bağlandığını kontrol etmekdir.

Assertion özelliğini kullanmak – İkinci Yol

```
assert ifade_1 : ifade_2;
```



Yukarıdaki ifadenin anlamı şudur: Eğer **ifade_1** false değeri geri dönerse, **ifade_2** deki değeri hata olarak fırlat.

Örnek - İkinci Yol

```
public class OrnekSinif {  
    public void ornekMetod( int parametre ) {  
        Asinifi a = null;  
        //.. Asinifi tipinde bir nesne almaya calisiliyor  
        // Islem basarili mi basarisiz mi olmus ?  
        assert a != null : "assert a != null : Nesne elde edilemedi, parametre=" + "parametre" ;  
    }  
}
```

- Bu gösterimimizde **a** referansı eğer Azinifi tipinde bir nesneye bağlanmamış ise yeni bir AssertionError tipinde bir hata fırlatılacaktır.
- Yanlız buradaki fark, bu AssertionError sınıfının yapılandırıcısına bizim bazı bilgiler gönderiyor olmamızdır.

Assertion ve derleme (*compile*) - I

- Assertion özelliği Java programlama diline yeni bir anahtar kelime kazandırmıştır.
- Java programlama dili daha evvelden yazılmış diğer uygulamalar için bir tehlike oluşturabilir.
- Bu tehlike geriye doğru uyumluluğun kalkması (*backwards compatibility*) yönündedir.

Assertion ve derleme (*compile*) - II

```
public class Uyum {  
  
    public static void main(String args[]) {  
  
        String assert = args[0] ; // dikkat  
        for (int i=0; i<10; i++) {  
            System.out.println( i + ":" + assert);  
        }  
    }  
}
```

- Yukarıdaki örneğimiz henüz assertion özelliği ortalarda yokken yazılmış olsun.
- Bu uygulamamızda, kullanıcıdan gelen ilk değeri String tipinde olan ve **assert** isimli bir referansa bağlanmaktadır.
- Yazılan Java kodlarının içerisinde assert anahtar kelimesi referans adı olarak geçiyorsa ve assertion özelliğini kullanmak istemiyorsanız, kısacası ben eski usul çalışmak istiyorum diyorsanız bazı ayrıntılara dikkat etmeniz gereklidir.

Assertion ve derleme (*compile*) - III

```
public class Uyum {  
  
    public static void main(String args[]) {  
  
        String assert = args[0] ; // dikkat  
        for (int i=0; i<10; i++) {  
            System.out.println( i + ":" + assert);  
        }  
    }  
}
```

Assertion özelliğini kullanmamak için...

```
> javac -source 1.3
```

Assertion özelliğini kullanmak için...

```
> javac -source 1.4
```

Assertion özelliğini nasıl kontrol ederim ? - I

- Assertion özelliğinin kıymetli kıalan en önemli faktör, bu özelliğin çalışma esnasında kapatılıp açılabiliyor olmasıdır.
- Örneğin bir uygulamanın geliştirilmesi esnasında assertion özelliği açık tutulabilir.
- Tahmin edilebileceği üzere assertion özelliğin açık tutulması belli bir performans kaybına sebebiyet verecektir.
- Fakat uygulamanın gelişimi tamamlandığı zaman assertion özelliği çalışma anında kapıtlarak (biraz sonra gösterilecek) bu performans kaybı engellenmiş olur.

Assertion özelliğini nasıl kontrol ederim ? - II

```
public class AssertTestBir {  
    public static void main(String[] args) {  
        assert 1 == 10;  
    }  
}
```

- *AssertTestBir.java* uygulamasını *javac -source 1.4* komutu ile derledikten sonra aşağıdaki gibi çalıştırılırsa...

```
> java AssertTestBir
```



```
> java -ea AssertTestBir
```

Assertion özelliğini nasıl kontrol ederim ? - III

```
public class AssertTestIki {  
  
    public static void main(String[] args) {  
        int gelDeger = ciftSayiAl(Integer.parseInt(args[0]));  
        assert gelDeger % 2 == 0; // dikkat  
    }  
  
    public static int ciftSayiAl(int sayi) {  
        return sayi * 3;  
    }  
}
```

Önce derleme (*compile*) aşaması

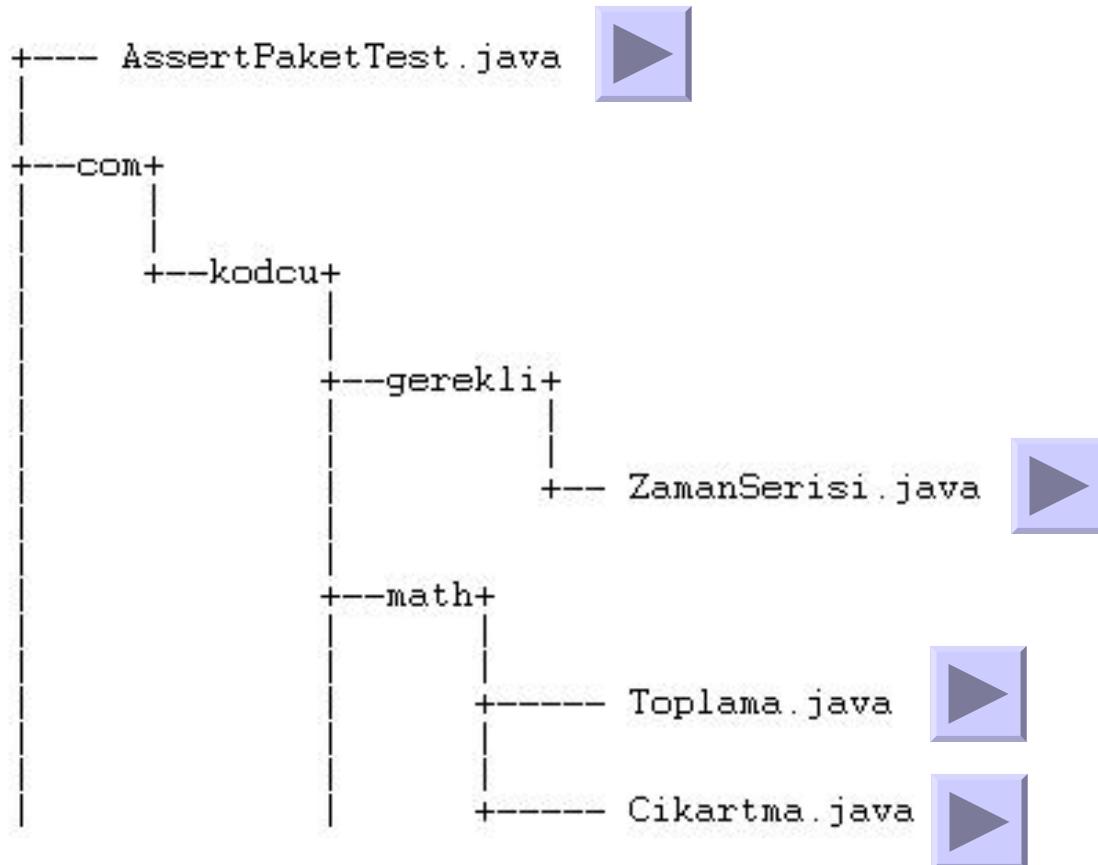
> **javac -source 1.4 AssertTestIki.java**

Sonra çalışma (*run*) aşaması

> **javac -ea AssertTestIki 5**

Paket kontrolleri

- Şimdi aşağıdaki gibi bir yapımız olsun.



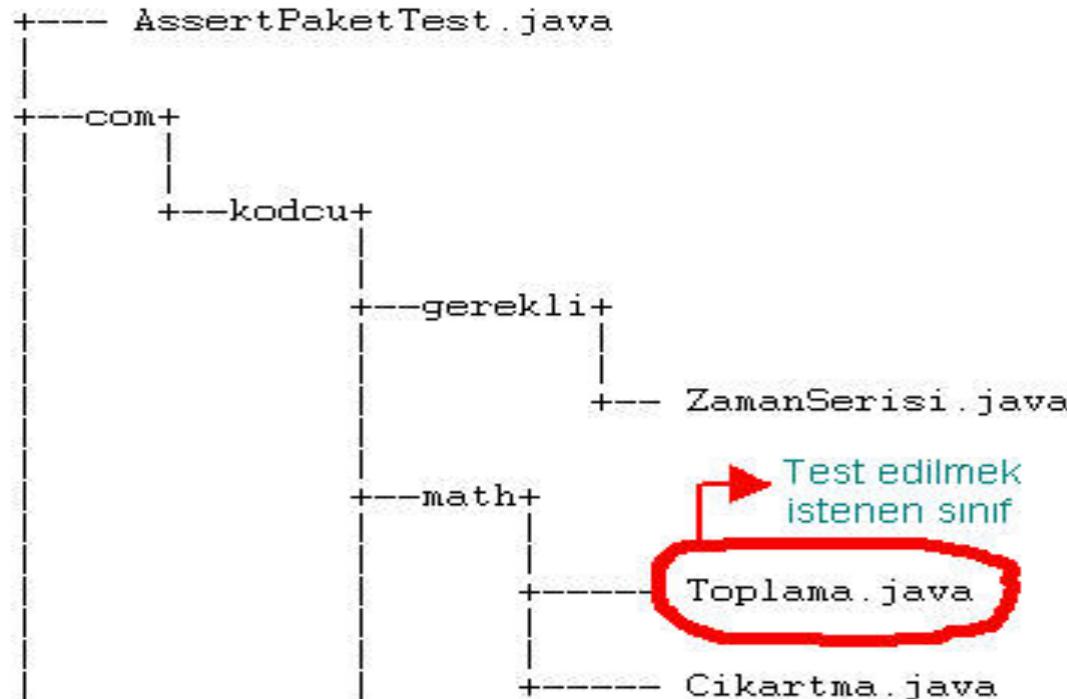
Tüm paketler için assertion özelliği açık

- AssertPaketTest sınıfının içerisindeki tüm paket ve bunlar içerisindeki sınıflar için assertion özelliğinin açık (etkin) olması isteniyorsa aşağıdaki komutun yazılması yeterli olacaktır.

```
> java -ea AssertPaketTest
```

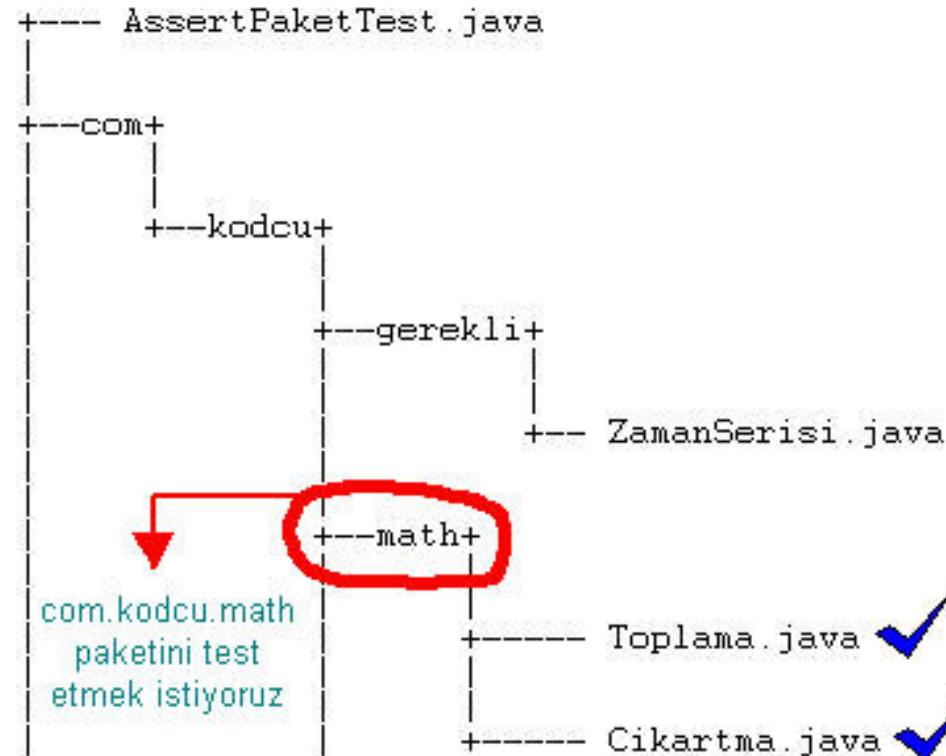
Toplama işlemlerinde bir hata var sanki....

- Assertion özelliğinin sadece *com.kodcu.math.Toplama* sınıfı için açılmak istenirse.



```
> java -ea:com.kodcu.math.Toplama AssertPaketTest
```

Sadece `com.kodcu.math.*` altındaki tüm sınıflar için assertion özelliği açmak istersek....



> `java -ea:com.kodcu.math AssertPaketTest`

> `java -ea:com.kodcu.math... AssertPaketTest`

Assertion özelliği açık mı ? Kapalı mı ?

```
package com.kodcu.gerekli;

public class ZamanSerisi {

    public ZamanSerisi() {

        boolean assertionEtkinMi = false;

        assert(assertionEtkinMi = true) : 

            if (assertionEtkinMi) {
                System.out.println("com.kodcu.gerekli.ZamanSerisi sınıfı"+
                    " için Assertion etkin");
            }
    }
}
```

Yukarıdaki sınıfımızın yapılandırıcısındaki assert anahtar kelimesinin olduğu satırı yakından bakacak olursak, burada bir karşılaştırma değil bir atama olduğunu görüyoruz.

AssertionError istisnalarını yakalamak

```
public class AssertionHataYakalama {  
    public static void main(String args[]) {  
        try {  
            int i = 10 ;  
            assert i == 20 ; // kesin assertion hatasi fırlatılacak  
        } catch (AssertionError ae) {  
            StackTraceElement ste[] = ae.getStackTrace();  
            if (ste.length>0) {  
                StackTraceElement sonuc = ste[0];  
                System.out.println( "Assertion hatasi : "+  
                    sonuc.getFileName()+"\n satir :" + sonuc.getLineNumber() );  
            } else {  
                System.out.println( "Herhangi bir bilgi yok" );  
            }  
            throw ae;// çok önemli  
        }  
    }  
}
```

Kural : Komut satırından girilmiş olan verilerin kontrolü için assertion özelliği kullanılmamalıdır.

- Assertion özelliği, uygulamanın kendi içerisinde tutarlığını sağlamak için kullanılmalıdır; kullanıcının uygulama ile olan tutarlığını sağlamak için değil.

```
public class KuralBirHatali {  
    static public void main( String args[] ) {  
  
        assert args.length == 3; // Yanlis kullanım  
        double x = Double.parseDouble( args[0] );  
        double y = Double.parseDouble( args[1] );  
        double z = Double.parseDouble( args[2] );  
    }  
}
```

Kural : Assertion özelliği, if (koşul)..... yerine kullanılmamalıdır.

- Kritik nokta, çalışma anında assertion özelliğinin kapatılması ile göz ardı edilir.

```
public class KuralIkiHatali {  
  
    private int port;  
    public void dinle() {  
        // Yanlış Kullanım  
        assert port > 1024 : "Bu port numarası kullanılamaz "+ port;  
        // ...  
    }  
}
```

Kural : **public** erişim belirliyicisine sahip olan yordamlara gönderilen parametreleri düz şekilde kontrol etmek amacıyla assertion özelliği kullanılmamalıdır.

```
public class KuralUcHatali {  
  
    public double varyansHesapla( double[] dizi ) {  
        assert dizi != null ; // Yanlis Kullanim  
        // ...  
    }  
    // ...  
}
```

Kural : Kullanıcıdan gelen verilerin mantık çerçevesinde olup olmadığı assertion özelliği ile kontrol edilmemelidir.

```
public void telefonNumarasiniAl( String telefonNumarasi ) {  
    if ( telefonNumarasi.length() == 7 ) {  
        // ...  
    } else if ( telefonNumarasi.length() == 11 ) {  
        // ...  
    } else {  
        // Yanlis Kullanim  
        assert false : "telefon numarasi 7 veya 11 haneden olusmali";  
    }  
}
```

Kural : Uygulamanın genel akışında assertion özelliğinin bir rolü olmamalıdır.

```
public class Test {  
  
    static private boolean hatayıSakla(Exception ex) {  
        // Hatayı saklama işlemi  
        // ...  
        return true;  
    }  
    static public void main( String args[] ) throws Exception {  
        // ...  
        try {  
            // ...  
        } catch (Exception ex) {  
            // Yanlış Kullanım  
            assert hatayıSakla(ex);  
        }  
    }  
}
```

Kural : private erişim belirleyicisine sahip olan yordamlara gönderilen parametrelerin kontrolünde assertion özelliği kullanılabilir.

- **private** erişim belirleyicisine sahip olan yordamlar dışarıdan ulaşılamaz.
- Bu tip yordamlar işlerin esas yapıldığı ve yanlış parametre gelmesinin affedilemeyeceği yerlerdir.

```
private double standartSapmayiBul( double[] dizi ) {  
    assert dizi != null ;  
}
```

Kural : Olmaz ise olmaz durumlarını yakalamak için assertion özelliği kullanılabilir.

```
private void hazirla() throws IOException {  
  
    Properties p = new Properties();  
    BufferedInputStream okuyucu = new BufferedInputStream(  
        new FileInputStream("c:\\gerekli\\bilgiler\\klc.properties"));  
  
    p.load(okuyucu);  
  
    kullaniciAdi = (String) p.get("KULLANACIADI");  
    sifre = (String) p.get("SIFRE");  
  
    assert kullaniciAdi != null; // dogru bir yaklasim  
    assert sifre != null; // dogru bir yaklasim  
  
}
```

Sorular ...

