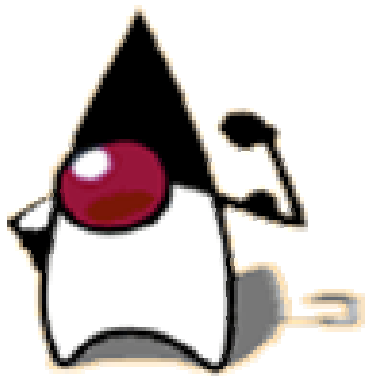
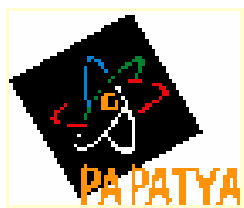
Başlangıç Durumuna Getirme ve Temizlik





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 yapılandırıcılar

- 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) farklıdırlar.
- Yapılandırıcılar, Java tarafından otomatik olarak çağrılırlar.
- Karşımıza çıkan iki problem
  - *Java Yapılandırıcının ismini nasıl bilecektir ?*
  - *Yapılandırıcının 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 içerisinden herhangi bir şekilde **`return`** ifadesi ile değer döndürülemez. (`return 5, return true` gibi)
- Yapılandırıcılardan 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 gerekir ?

Örnek - 1 (devam)

Yordam isimleri olarak

- muzik dosyası için **muzikDosyasiAc()**
- resim dosyası için **resimDosyasıAc()**
- text dosyası için **textDosyasıAc()**

Örnek - 1 (devam)

- Sonuçta işlem sadece dosya açmaktır, 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 ayırt edilebilir mi ?

- Akıllara şöyle bir soru gelebilir : "Adaş yordamlar dönüş tiplerine göre ayırt 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 yapılandırcılar (parametresiz yapılandırcılar, default constructor veya "no-args" constructor) içi boş bir yordam olarak düşünülebilir.

Örnek - 2

```
class Kedi {
  int i;
}

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

Örnek - 2 (devam)

```
class Kedi {
  int i;
  /* varsayılan yıpılandırıcı.
      Bu yapılandırıcıyı eğer biz koymasaydık
      Java bizim yerimize zaten koyardı
 */
  public Kedi() {}
}

public class VarsayilanYapilandirici {
  public static void main(String[] args) {
    // varsayılan yıpılandırıcı
     Kedi kd = new Kedi();
  }
}
```

Büyünün Bozulması

- Eğer kendimiz yapılandırıcı yazarsak, Java bizden varsıyılan yapılandırıcı desteğini çekecektir.

- Kendimize ait özel yapılandırıcılar tanımlarsak Java'ya ***"Ben ne yaptığımı biliyorum, lütfen karışma"*** demiş oluruz.

VarsayilanYapilandiriciVersiyon2.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 cağrımlarında `this` kullanımı - 2

Yumurta.java

Bir yapılandırıcıdan diğerini çağırmak

- Yapılandırıcı içerisinden diğer bir yapılandırıcıyı çağırırken **`this`** ifadesi her zaman ilk satırda yazılmalıdır.
- Her zaman yapılandırıcılar içerisinden **`this`** ifadesi ile başka bir yapılandırıcı çağrılır.
- Yapılandırıcılar içersinde birden fazla **`this`** ifadesi ile başka yapılandırıcı çağrılamaz.

Tost.java

Ekran çıktısı

Tost(int sayi ,String malzeme)

Tost(int sayi)

parametresiz yapilandirici

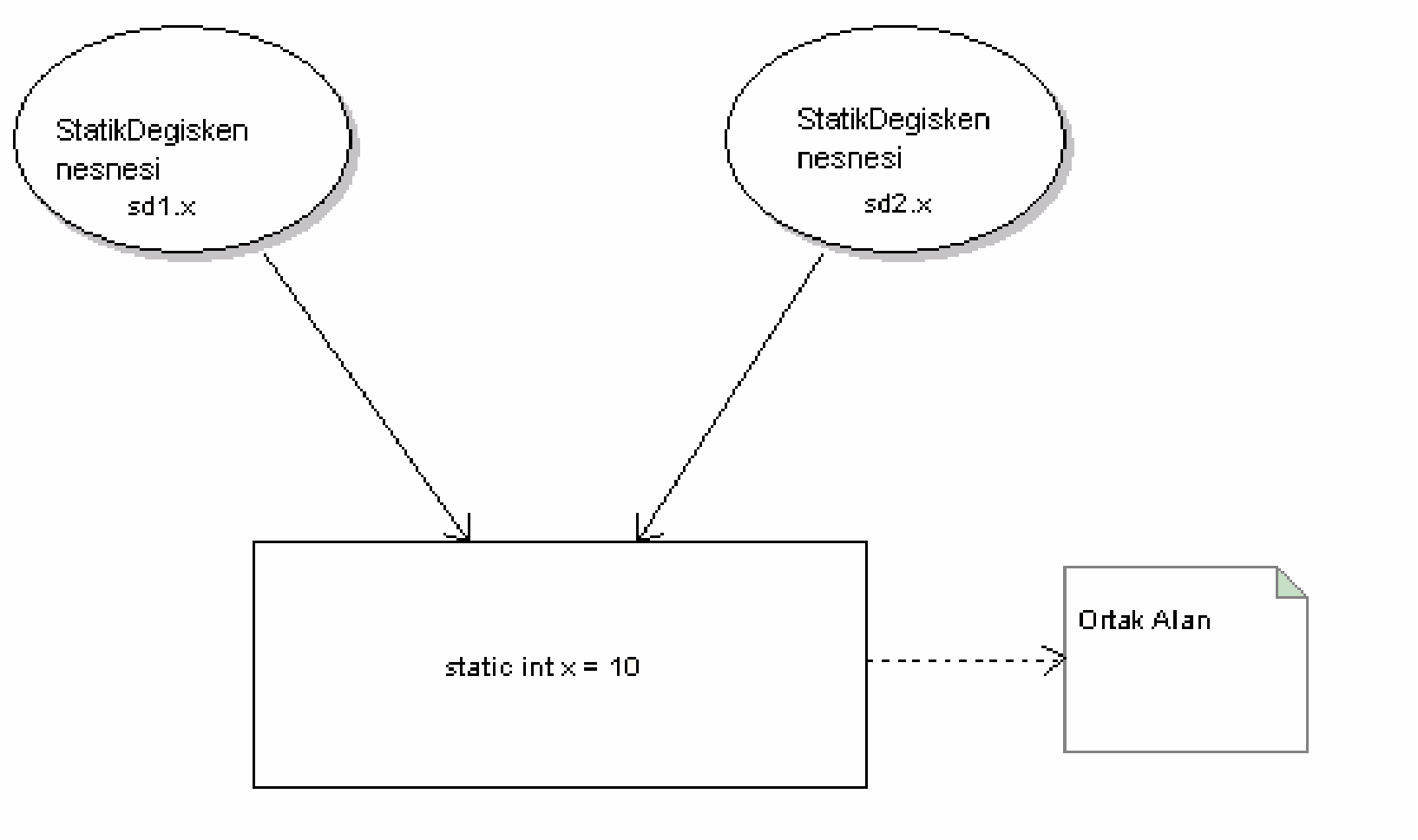
Tost sayisi =5 malzeme =Sucuklu

Statik Alanlar (Sınıf Alanları)

- Sadece global olan alanlara statik özelliğini verebiliriz.
- Yerel değişkenlerin statik olma özellikleri yoktur.
- Statik alanlar, bir sınıfa 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 çağırmak için herhangi bir sınıfa ait nesne oluşturma zorunluluğu yoktur.
- Statik olmayan yordamlardan (nesneye ait yordamlar), statik yordamları rahatlıkla çağırılabilmesine karşın statik yordamlardan nesne yordamlarını doğrudan çağıramayız.

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 toplayıcısı tetiklendi

Çöp toplayıcısı , gereksiz 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.

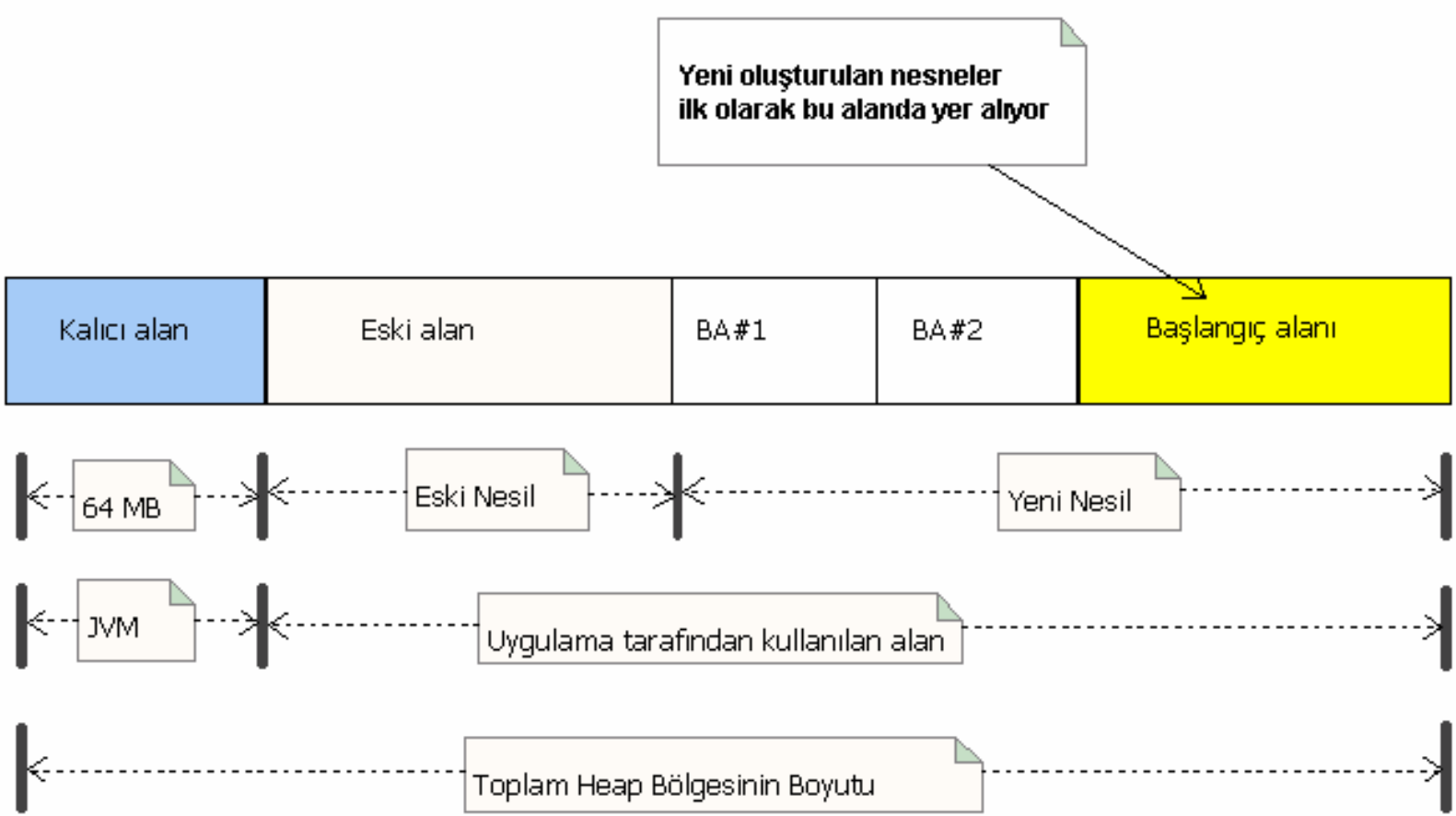
- Bu görev çöp toplayıcısına aittir. Java 1.3.1 ve daha sonraki Java versiyonları iki noktayı garanti eder;

  – 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.

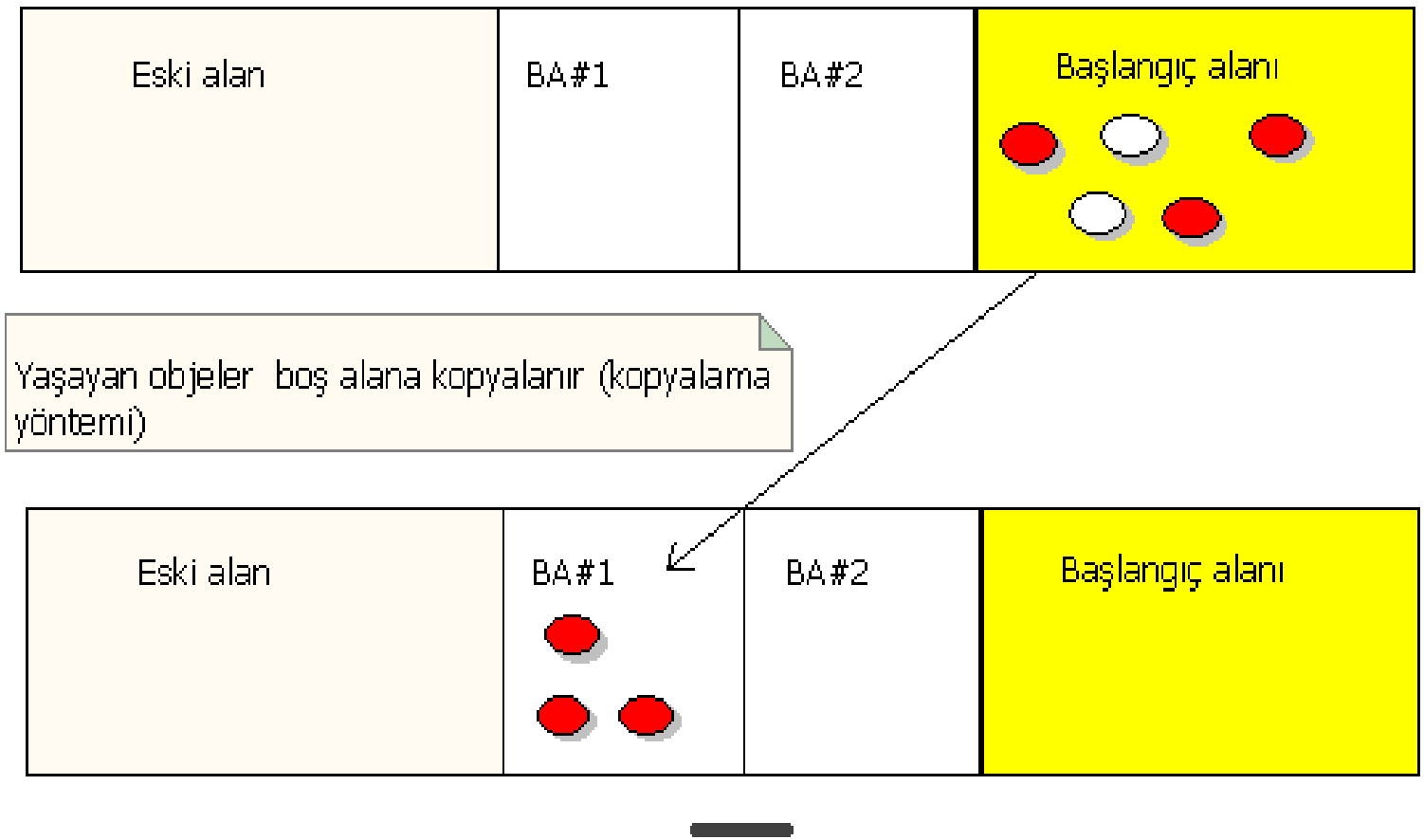
- Heap bölgesine minimum veya maksimum değerleri vermek için **-Xms** veya **-Xmx** parametlerini kullanırız.

```
java -Xms32mb Temizle
```

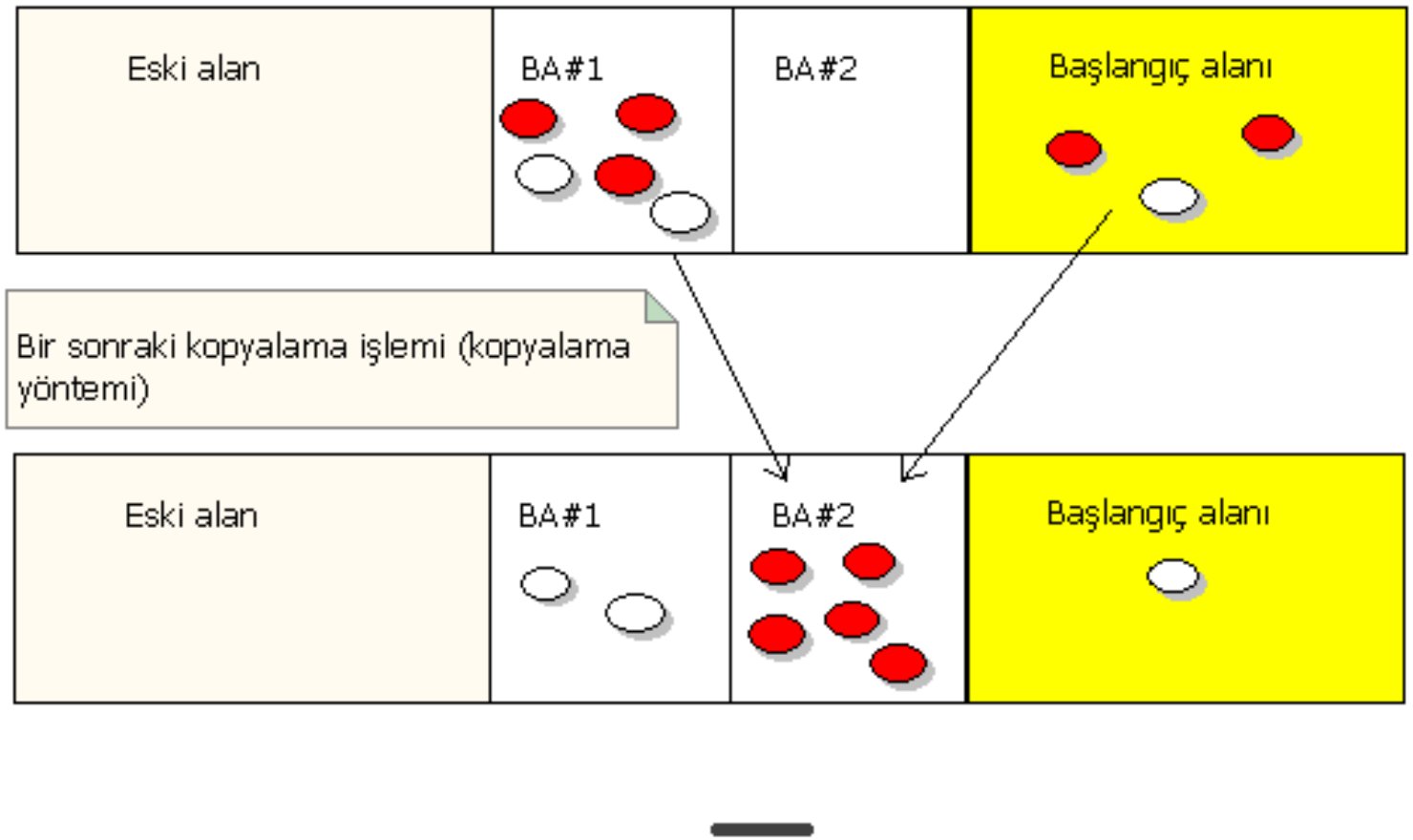
Kopyalama yönteminin gösterimi

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

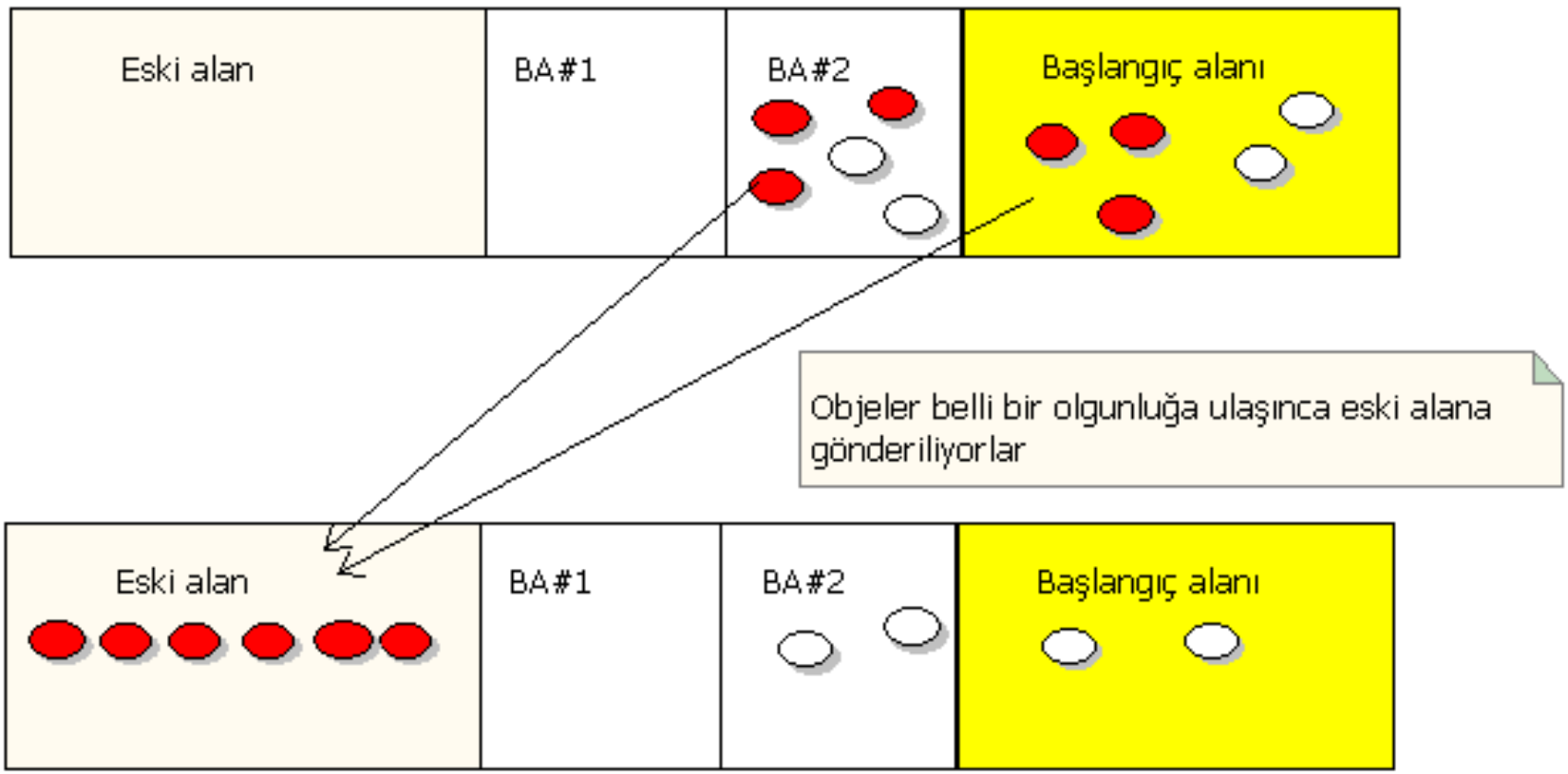
Aşama – 1


Eski alan
BA#1
BA#2
Başlangıç alanı
Yaşayan objeler boş alana kopyalanır (kopyalama yöntemi)
Eski alan
BA#1
BA#2
Başlangıç alanı

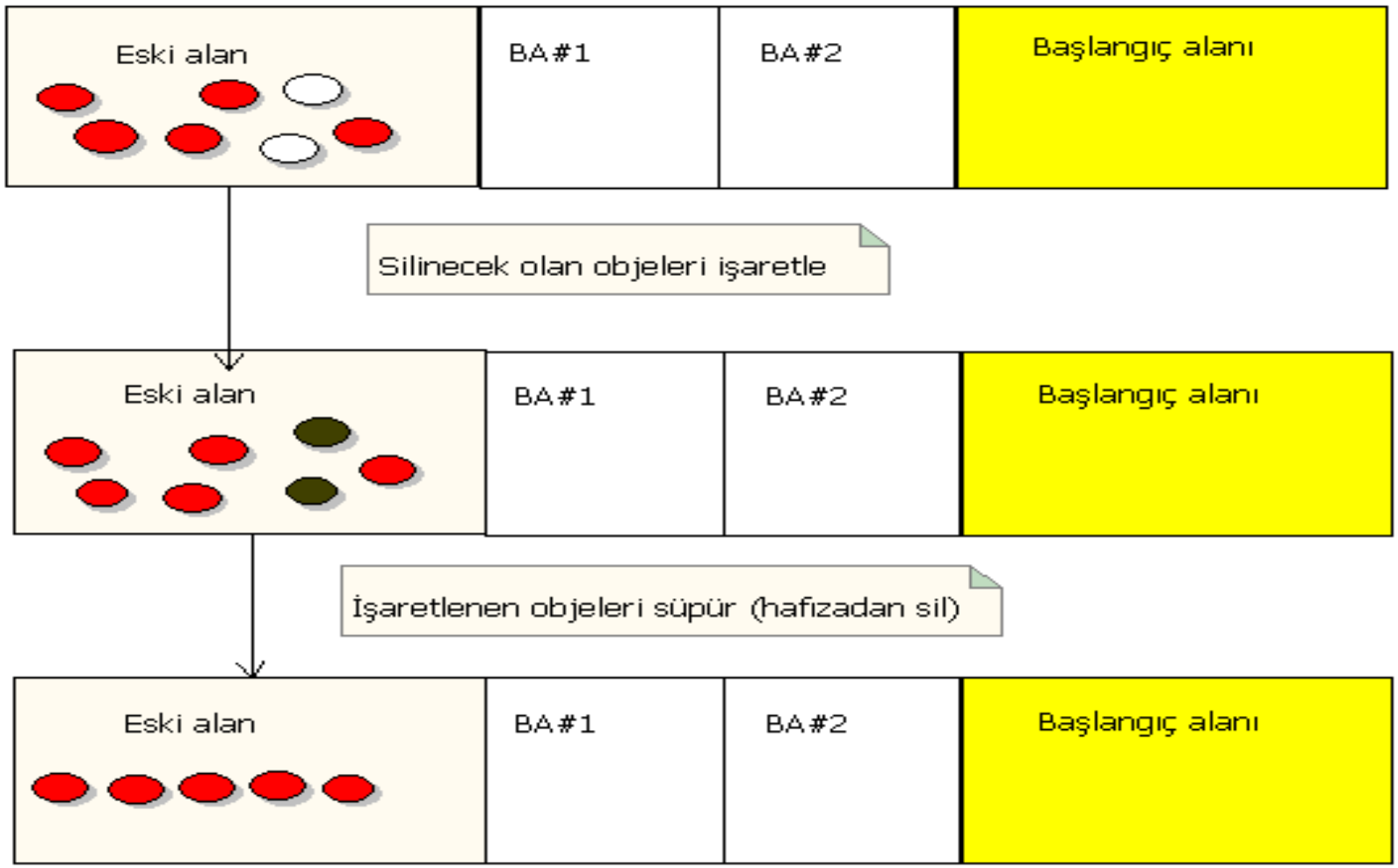
Aşama – 2


Eski alan
BA#1
BA#2
Başlangıç alanı
Bir sonraki kopyalama işlemi (kopyalama yöntemi)
Eski alan
BA#1
BA#2
Başlangıç alanı

Aşama – 3


Eski alan
BA#1
BA#2
Başlangıç alanı
Objeler belli bir olgunluğa ulaşınca eski alana gönderiliyorlar
Eski alan
BA#1
BA#2
Başlangıç alanı

İşaretle ve süpür yönteminin gösterimi

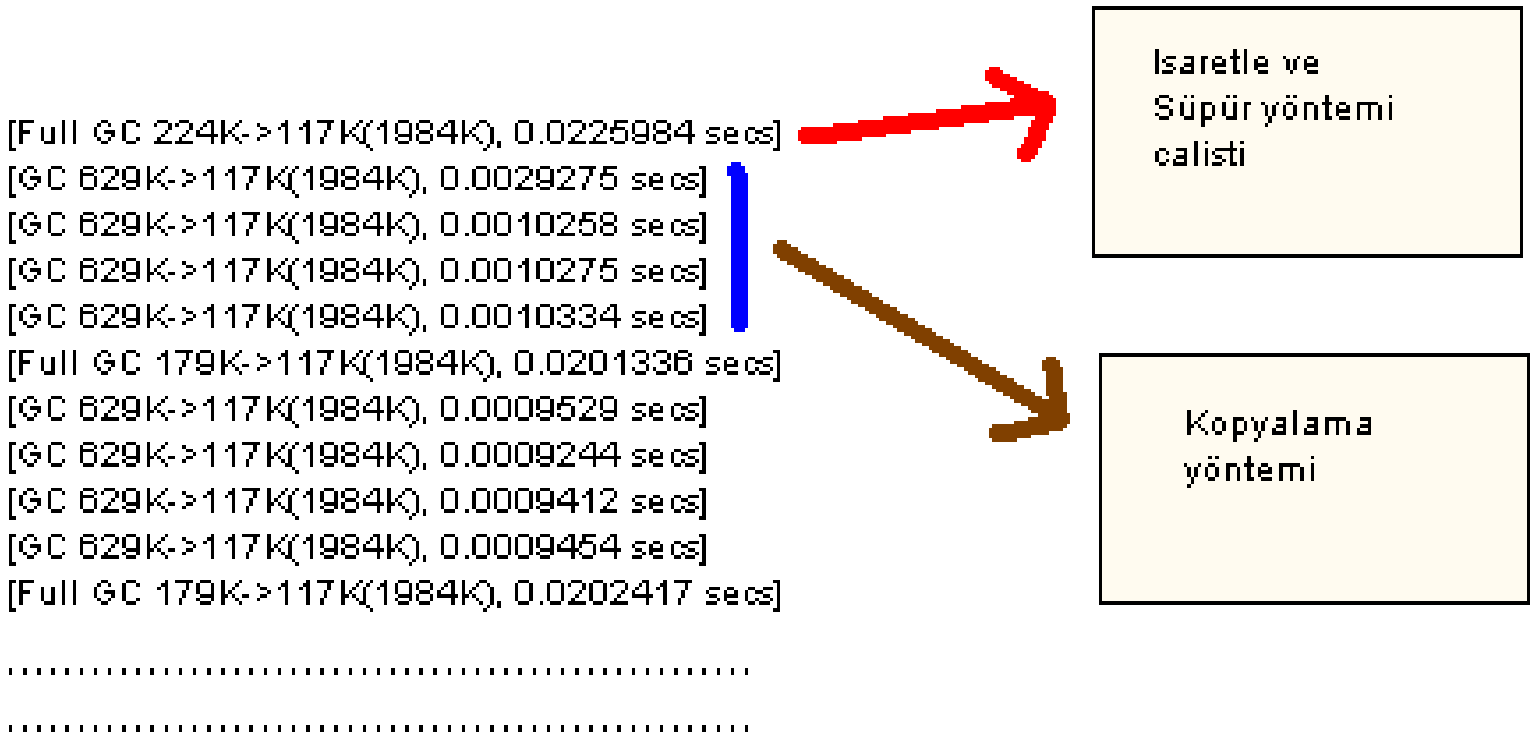


Kopyala yöntemin ve işaretle ve süpür yöntemi

HeapGosterim.java

```
java -verbosegc HeapGosterim
```

Uygulamanın Çıktısı



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

DegiskenGosterim.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.

IlkelTiplerStatik.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

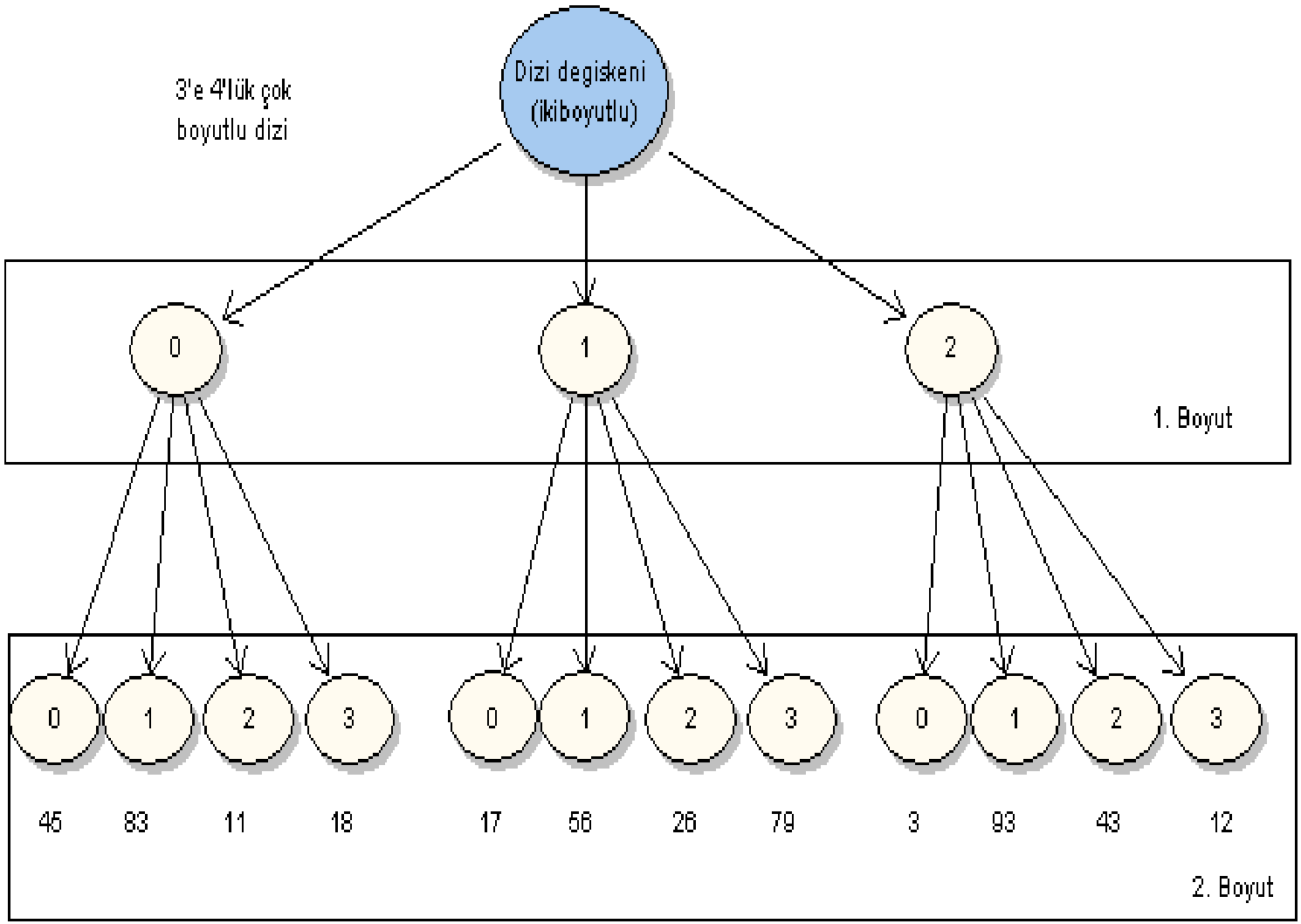
Dizi içerisinde dizi tanımlanabilir.

```
int[][] t1 = {

      { 1, 2, 3, },

      { 4, 5, 6, },

   };
```

Çok boyutlu dizileri oluşturmanın diğer bir yolu

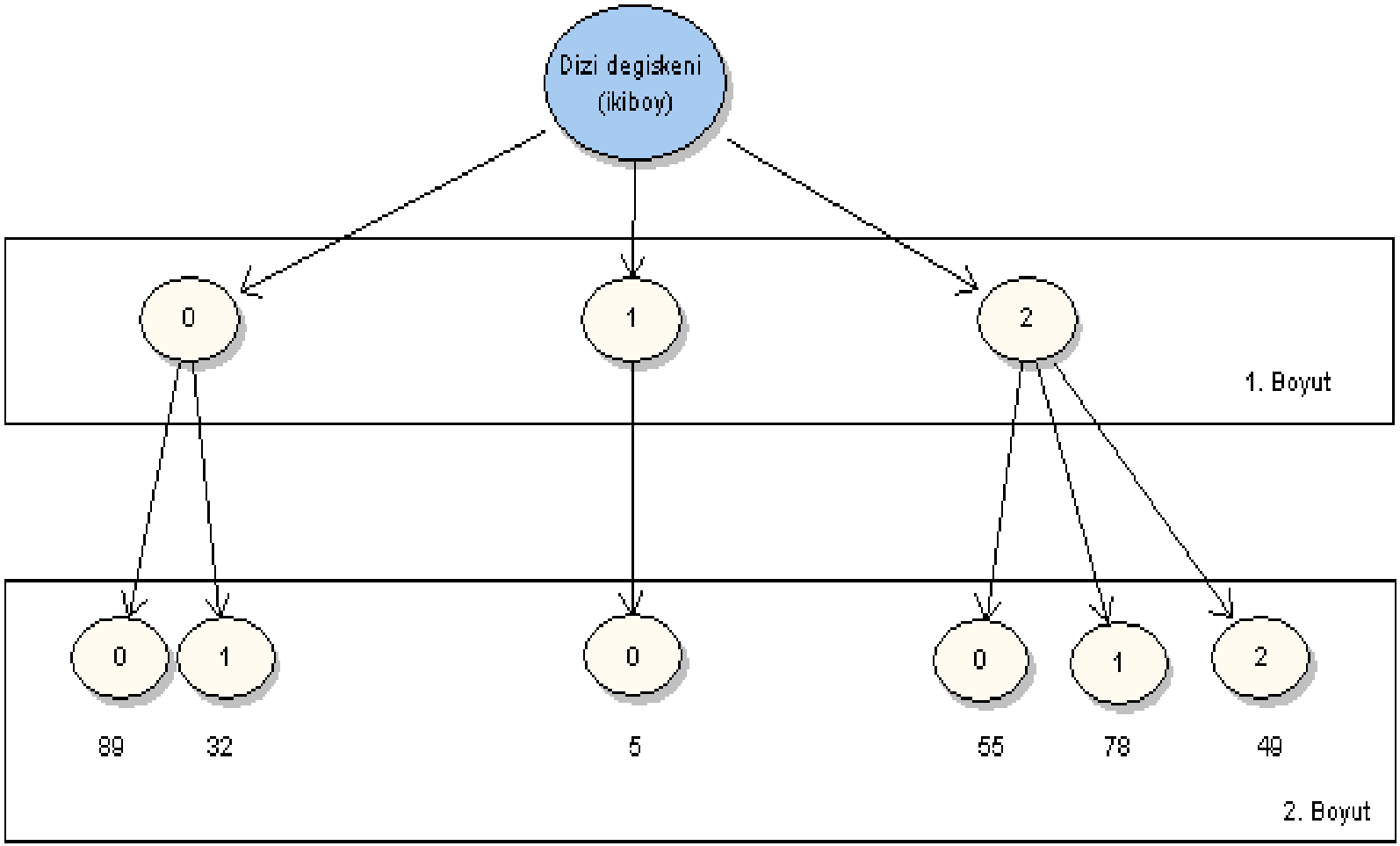
```
int [][] t1 = new int [3][4] ;

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

CokBoyutluDizilerOrnekBir.java


3'e 4'lük çok
boyutlu dizi
Dizi degiskeni
(ikiboyutlu)
0
1
2
1. Boyut
0
1
2
3
0
1
2
3
0
1
2
3
45
83
11
18
17
56
26
79
3
93
43
12
2. Boyut

Değişik boyuta sahip diziler

CokBoyutluDiziler.java


Dizi degiskeni
(ikiboy)
0
1
2
1. Boyut
0
1
0
0
1
2
89
32
5
55
78
49
2. Boyut

Sorular ...