

Bölüm 5: Diziler

Java ile Nesne Yönelimli Programlama

Dizi kavramı ve indis

Tanımlama ve ilk değer verme

Referanslar ve bellek

Ortalama, ortanca, mod

Arama ve eşleştirme

Seçimli sıralama

Çok boyutlu diziler

Matris transpozu

Tic Tac Toe

Sercan KÜLCÜ · Giresun Üniversitesi · Elektrik-Elektronik Mühendisliği

Öğrenme Çıktıları

Kavram

Dizinin (array) aynı tipten, sabit sayıda öğeyi ardışık hücrelerde tuttuğunu ve öğelere 0'dan başlayan **indis** ile erişildiğini açıklamak.

Bellek

Dizi değişkeninin bir **referans** olduğunu, dizinin heap'te durduğunu; atama, kopyalama ve metoda geçirmenin sonucunu öngörmek.

Algoritmalar

Ortalama, ortanca, mod, doğrusal arama, eşleştirme, tersine çevirme ve seçimli sıralamayı yazmak; $O(1)$, $O(n)$, $O(n^2)$ maliyetlerini karşılaştırmak.

Söz dizimi

Tek ve çok boyutlu dizileri tanımlamak, **new** ve ilkendirici listesiyle oluşturmak, **length** ile dolaşmak.

Hatalar

ArrayIndexOutOfBoundsException ve sık derleme hatalarını tanımak, sınır kontrolüyle önlemek.

Uygulama

Matrisleri iç içe döngülerle doldurmak, yazdırmak, transpozunu almak ve **char[][]** ile Tic Tac Toe oyunu kurmak.

Dizi Nedir?

- **Dizi (Array):** aynı **tipteki** verileri düzenli bir şekilde, tek bir ad altında saklayan yapıdır.
- Aynı adla tanımlanan, bir veya daha fazla veri saklayabilen bir değişken gibi düşünülebilir.
- Öğelere **indis (index)** numarasıyla erişilir; indisleme sayesinde erişim çok hızlıdır.
- **İlk öğenin indisi 0**'dır; n öğeli dizinin son indisi $n - 1$ 'dir.
- Bir öğeye erişmek için dizinin adı ve köşeli parantez içinde indis yazılır: `dizi[0]` .

Örnek: `dizi = [10, 20, 30, 40, 50]` ise `dizi[0]` ilk öğedir ve değeri **10**'dur; `dizi[4]` son öğedir (50).

Apartman daireleri

Tek bir adres (dizinin adı) ve numaralı daireler (indisler). "3 numara" demek, daireyi bulmaya yeter.

Yumurta kolisi

Hücre sayısı baştan bellidir; koli büyümmez. Her hücreye yalnızca aynı tip şey konur.

DİZİ KAVRAMI

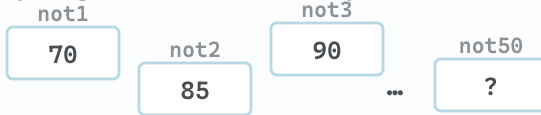
Neden Dizi?

DİZİSİZ: 50 ÖĞRENCİ İÇİN 50 DEĞİŞKEN

```
int not1 = 70;  
int not2 = 85;  
int not3 = 90;  
// ... not50'ye kadar  
int toplam = not1 + not2 + not3 /* + ... */;
```

Ortalamayı hesaplamak için 50 ismi tek tek yazmak gerekir; öğrenci sayısı değişirse kod da değişir.

Ayrı değişkenler: her birinin adı ayrı

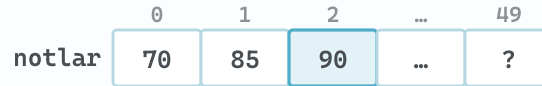


DİZİYLE: TEK AD, DÖNGÜYLE İŞLEM

```
int[] notlar = new int[50];  
// ... notlar doldurulur  
int toplam = 0;  
for (int i = 0; i < notlar.length; i++) {  
    toplam += notlar[i];  
}
```

Aynı kod 5 ya da 5000 not için çalışır: **indis bir değişken olabilir** (`notlar[i]`), döngü tüm öğeleri dolaşır.

Dizi: tek ad, numaralı hücreler



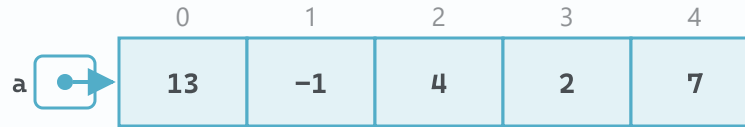
▲ `notlar[i]`, `i = 2`

Dizinin gücü buradadır: `notlar[i]` yazıldığında hangi hücreye bakılacağı **çalışma zamanında** `i` 'nin değerine göre belirlenir.

DİZİ KAVRAMI

Bellekte Bir Dizi

Aşağıdaki görsel, beş ögeli tek boyutlu bir diziyi temsil eder. Her öge bir **hücrede** durur; hücreler bellekte **yan yana (ardışık)** yerleşir.



indis	0	1	2	3	4
öge	13	-1	4	2	7

`a[1]` = -1, `a[4]` = 7, `a.length` = 5. `a[5]` diye bir hücre **yoktur**: geçerli indisler 0...4'tür.

Avantajlar ve Dezavantajlar

Avantajları

- + Verileri **düzenli ve ardışık** bir şekilde saklar.
- + İndisleme ile öğelere **hızlı erişim** sağlar: `dizi[i]` her `i` için aynı sürede okunur.
- + Verileri gruplandırarak düzenlemeye olanak sağlar; döngülerle kolayca işlenir.
- + Bellek kullanımı yalındır: yalnızca öğeler ve uzunluk tutulur.

Dezavantajları

- **Sabit boyutludur**: oluşturulduktan sonra boyutu değiştirilemez.
- Yer kalmadığında öğe eklemek için **daha büyük yeni bir dizi** oluşturup kopyalamak gerekir.
- Ortaya ekleme veya ortadan silme, öğeleri **kaydırmayı** gerektirir (ek maliyet).
- Yalnızca tek tipten öğe tutar.

Boyutu değişen listeler için Java'da `ArrayList` vardır; içinde yine bir dizi kullanır ve dolunca büyütür (sonraki bölüm: Koleksiyonlar).



Dizi Türleri ve Kullanım Alanları

Tek boyutlu

Öğeler tek bir satırda saklanır.

`int[]` notlar

İki boyutlu

Öğeler satır ve sütunlarda saklanır (matris).

`int[][]` tablo

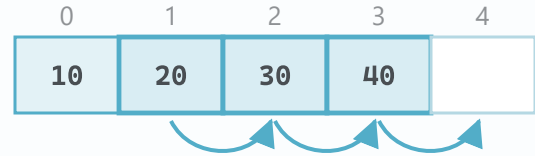
Çok boyutlu

Daha karmaşık düzeyde veri saklama ve erişimi destekler.

`int[][][]` kup

KULLANIM ALANLARI

- Programlama dillerinde en yaygın veri yapısıdır; verileri düzenlemek ve işlemek için idealdir.
- **Ölçümler:** bir sensörden saniyede alınan örnekler, ses sinyali (`double[]`).
- **Matrisler:** devre analizi, doğrusal denklem sistemleri.
- **Resimler:** piksel ızgarası (`int[yukseklik][genislik]`).



15'i indis 1'e eklemek: önce 40, 30, 20 birer sağa kayar

Dizi Tanımlama

```
VeriTipi[] diziAdi = new VeriTipi[diziBoyutu];
```

VeriTipi

Dizide saklanacak verilerin tipi:
`int`, `double`, `char`, `String`...

diziAdi

Dizinin adı; değişken adlandırma kurallarına uyar.

diziBoyutu

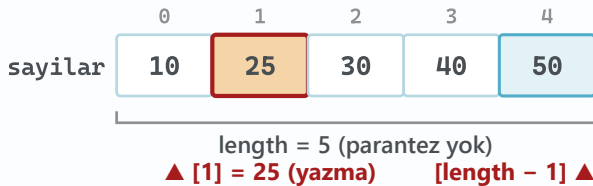
Dizinin kaç öge içereceği; bir sabit ya da çalışma zamanında hesaplanan bir `int`.

```
int[] sayilar = new int[5];           // 5 tamsayı
double[] olcum = new double[100];     // 100 ondalık
String[] isimler = new String[30];
int n = 12;
char[] harfler = new char[n];         // boyut değişkenden
int dizi2[] = new int[3];             // C tarzı, geçerli ama önerilmez
```

Tip	Varsayılan değer
<code>int</code> , <code>long</code> ...	<code>0</code>
<code>double</code>	<code>0.0</code>
<code>boolean</code>	<code>false</code>
<code>char</code>	<code>'\u0000'</code> (kod 0)
<code>String</code> ve nesneler	<code>null</code>

Erişim ve length

- Dizi öğelerine **indis** ile erişilir; indis 0'dan başlar.
- `dizi[i]` hem **okumak** hem **yazmak** için kullanılır.
- Dizinin boyutu `length` özelliği ile alınır: dizinin kaç öğe içerdiğini verir.
- `length` bir metod değildir: **parantez yok** (`String` 'deki `length()` ile karıştırmayın).
- Son öğe her zaman `dizi[dizi.length - 1]` 'dir.



```
int[] sayilar = {10, 20, 30, 40, 50};
int ilkEleman = sayilar[0];    // İlk elemanı alır
int ikinciEleman = sayilar[1]; // İkinci elemanı alır
int diziBoyutu = sayilar.length; // 5
System.out.println(ilkEleman + " " + ikinciEleman
    + " " + diziBoyutu);
sayilar[1] = 25;                // yazma
int son = sayilar[sayilar.length - 1]; // 50
```

ÇIKTI

10 20 5

Dizi İlk Değer Verme

- Dizi tanımlanırken öğelerine **başlangıç değeri** atanabilir: **iklendirici listesi** (array initializer).
- Bu durumda boyut **belirtilmez**; iklendiriciler boyutu belirler: 5 değer varsa dizi 5 öğeli olur.

```
int[] sayilar = {10, 20, 30, 40, 50}; // 5 elemanlı
int[] n = {1, 2, 3, 4, 5}; // 5 değer olduğundan n 5 elemanlı bir dizi olur
int[] m = new int[]{1, 2, 3}; // uzun biçim: her yerde kullanılabilir
String[] gunler = {"Pzt", "Sal", "Çar"};
char[] sesli = {'a', 'e', 'ı', 'i', 'o', 'ö', 'u', 'ü'};
```

Kısa biçim yalnızca tanımlamada

Sonradan `a = {1, 2, 3};` yazılamaz (illegal start of expression). Bunun yerine `a = new int[]{1, 2, 3};` yazılır.

İkisi birden olmaz

`new int[3]{1, 2, 3}` derlenmez: ya boyut ya liste verilir.

JAVA'DA DİZİLER

Örnek: DiziOrnegi

```
public class DiziOrnegi {  
    public static void main(String[] args) {  
        // Bir diziyi tanımlama: 5 elemanlı bir tamsayı dizisi  
        int[] sayilar = new int[5];  
        // Dizinin elemanlarına değer atama  
        sayilar[0] = 10;  
        sayilar[1] = 20;  
        sayilar[2] = 30;  
        sayilar[3] = 40;  
        sayilar[4] = 50;  
        // Dizinin elemanlarına erişme  
        System.out.println("birinci eleman: " + sayilar[0]);  
        System.out.println("ucuncu eleman: " + sayilar[2]);  
        // Dizinin tüm elemanlarını döngü içerisinde yazdırma  
        System.out.print("Elemanlar: ");  
        for (int i = 0; i < sayilar.length; i++) {  
            System.out.print(sayilar[i] + " ");  
        }  
    }  
}
```

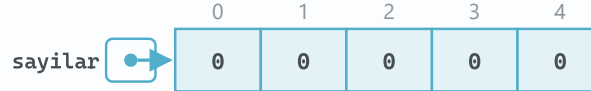
ÇIKTI

birinci eleman: 10
ucuncu eleman: 30
Elemanlar: 10 20 30 40 50

Döngü kalıbı

for (int i = 0; i < dizi.length; i++) dizinin **tüm** indislerini 0'dan length - 1'e kadar dolaşır. <= yazmak sık yapılan bir hatadır.

DiziOrnegi: Kodu Adım Adım İzle



`new int[5]` heap'te 5 hücrelik bir dizi oluşturur; her hücre **varsayılan değer 0** ile başlar. `sayilar` bu dizinin referansını tutar.

Adım 1 / 15

```
► int[] sayilar = new int[5];
2 sayilar[0] = 10;
3 sayilar[1] = 20;
4 sayilar[2] = 30;
5 sayilar[3] = 40;
6 sayilar[4] = 50;
7 System.out.println("birinci eleman: " + sayilar[0]);
8 System.out.println("ucuncu eleman: " + sayilar[2]);
9 System.out.print("Elemanlar: ");
10 for (int i = 0; i < sayilar.length; i++) {
11     System.out.print(sayilar[i] + " ");
12 }
```

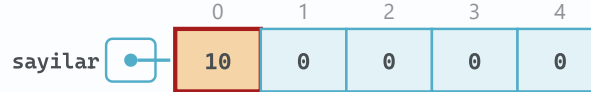
DEĞİŞKENLER

`sayilar = [0, 0, 0, 0, 0]`

ÇIKTI



DiziOrnegi: Kodu Adım Adım İzle



`sayilar[0] = 10;` → 1. hücreye (indis 0) yazılır.

Adım 2 / 15

```
1 int[] sayilar = new int[5];
2 sayilar[0] = 10;
3 sayilar[1] = 20;
4 sayilar[2] = 30;
5 sayilar[3] = 40;
6 sayilar[4] = 50;
7 System.out.println("birinci eleman: " + sayilar[0]);
8 System.out.println("ucuncu eleman: " + sayilar[2]);
9 System.out.print("Elemanlar: ");
10 for (int i = 0; i < sayilar.length; i++) {
11     System.out.print(sayilar[i] + " ");
12 }
```

DEĞİŞKENLER

`sayilar = [10, 0, 0, 0, 0]`

ÇIKTI

DiziÖrneği: Kodu Adım Adım İzle



`sayilar[1] = 20;` → 2. hücreye (indis 1) yazılır.

Adım 3 / 15

```
1 int[] sayilar = new int[5];
2 sayilar[0] = 10;
3 ► sayilar[1] = 20;
4 sayilar[2] = 30;
5 sayilar[3] = 40;
6 sayilar[4] = 50;
7 System.out.println("birinci eleman: " + sayilar[0]);
8 System.out.println("ucuncu eleman: " + sayilar[2]);
9 System.out.print("Elemanlar: ");
10 for (int i = 0; i < sayilar.length; i++) {
11     System.out.print(sayilar[i] + " ");
12 }
```

DEĞİŞKENLER

`sayilar = [10, 20, 0, 0, 0]`

ÇIKTI



DiziOrnegi: Kodu Adım Adım İzle



`sayilar[2] = 30;` → 3. hücreye (indis 2) yazılır.

Adım 4 / 15

```
1 int[] sayilar = new int[5];
2 sayilar[0] = 10;
3 sayilar[1] = 20;
4 ► sayilar[2] = 30;
5 sayilar[3] = 40;
6 sayilar[4] = 50;
7 System.out.println("birinci eleman: " + sayilar[0]);
8 System.out.println("ucuncu eleman: " + sayilar[2]);
9 System.out.print("Elemanlar: ");
10 for (int i = 0; i < sayilar.length; i++) {
11     System.out.print(sayilar[i] + " ");
12 }
```

DEĞİŞKENLER

`sayilar = [10, 20, 30, 0, 0]`

ÇIKTI

DiziOrnegi: Kodu Adım Adım İzle



`sayilar[3] = 40;` → 4. hücreye (indis 3) yazılır.

Adım 5 / 15

```
1 int[] sayilar = new int[5];
2 sayilar[0] = 10;
3 sayilar[1] = 20;
4 sayilar[2] = 30;
5 ► sayilar[3] = 40;
6 sayilar[4] = 50;
7 System.out.println("birinci eleman: " + sayilar[0]);
8 System.out.println("ucuncu eleman: " + sayilar[2]);
9 System.out.print("Elemanlar: ");
10 for (int i = 0; i < sayilar.length; i++) {
11     System.out.print(sayilar[i] + " ");
12 }
```

DEĞİŞKENLER

`sayilar = [10, 20, 30, 40, 0]`

ÇIKTI

DiziOrnegi: Kodu Adım Adım İzle



`sayilar[4] = 50;` → 5. hücreye (indis 4) yazılır.

Adım 6 / 15

```
1 int[] sayilar = new int[5];
2 sayilar[0] = 10;
3 sayilar[1] = 20;
4 sayilar[2] = 30;
5 sayilar[3] = 40;
6 ▶ sayilar[4] = 50;
7 System.out.println("birinci eleman: " + sayilar[0]);
8 System.out.println("ucuncu eleman: " + sayilar[2]);
9 System.out.print("Elemanlar: ");
10 for (int i = 0; i < sayilar.length; i++) {
11     System.out.print(sayilar[i] + " ");
12 }
```

DEĞİŞKENLER

`sayilar = [10, 20, 30, 40, 50]`

ÇIKTI

DiziOrnegi: Kodu Adım Adım İzle



Birinci öğenin indisi **0**'dir: `sayilar[0]` okunur → 10.

Adım 7 / 15

```
1 int[] sayilar = new int[5];
2 sayilar[0] = 10;
3 sayilar[1] = 20;
4 sayilar[2] = 30;
5 sayilar[3] = 40;
6 sayilar[4] = 50;
7 ▶ System.out.println("birinci eleman: " + sayilar[0]);
8 System.out.println("ucuncu eleman: " + sayilar[2]);
9 System.out.print("Elemanlar: ");
10 for (int i = 0; i < sayilar.length; i++) {
11     System.out.print(sayilar[i] + " ");
12 }
```

DEĞİŞKENLER

`sayilar = [10, 20, 30, 40, 50]`

ÇIKTI

birinci eleman: 10

DiziOrnegi: Kodu Adım Adım İzle



Üçüncü öge `sayilar[2]` 'dir (indis = sıra - 1) → 30.

Adım 8 / 15

```
1 int[] sayilar = new int[5];
2 sayilar[0] = 10;
3 sayilar[1] = 20;
4 sayilar[2] = 30;
5 sayilar[3] = 40;
6 sayilar[4] = 50;
7 System.out.println("birinci eleman: " + sayilar[0]);
8 System.out.println("ucuncu eleman: " + sayilar[2]);
9 System.out.print("Elemanlar: ");
10 for (int i = 0; i < sayilar.length; i++) {
11     System.out.print(sayilar[i] + " ");
12 }
```

DEĞİŞKENLER

`sayilar = [10, 20, 30, 40, 50]`

ÇIKTI

```
birinci eleman: 10
ucuncu eleman: 30
```

DiziOrnegi: Kodu Adım Adım İzle



`print` alt satıra geçmez; öğeler aynı satıra yazılacak.

Adım 9 / 15

```
1 int[] sayilar = new int[5];
2 sayilar[0] = 10;
3 sayilar[1] = 20;
4 sayilar[2] = 30;
5 sayilar[3] = 40;
6 sayilar[4] = 50;
7 System.out.println("birinci eleman: " + sayilar[0]);
8 System.out.println("ucuncu eleman: " + sayilar[2]);
9 System.out.print("Elemanlar: ");
10 for (int i = 0; i < sayilar.length; i++) {
11     System.out.print(sayilar[i] + " ");
12 }
```

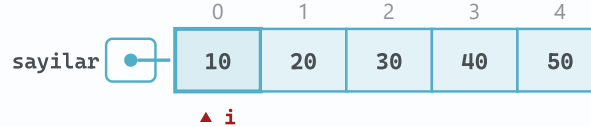
DEĞİŞKENLER

sayilar = [10, 20, 30, 40, 50]

ÇIKTI

```
birinci eleman: 10
ucuncu eleman: 30
Elemanlar:
```

DiziOrnegi: Kodu Adım Adım İzle



`i = 0` : `0 < 5` doğru → `sayilar[0]` = 10 yazılır.

Adım 10 / 15

```
1 int[] sayilar = new int[5];
2 sayilar[0] = 10;
3 sayilar[1] = 20;
4 sayilar[2] = 30;
5 sayilar[3] = 40;
6 sayilar[4] = 50;
7 System.out.println("birinci eleman: " + sayilar[0]);
8 System.out.println("ucuncu eleman: " + sayilar[2]);
9 System.out.print("Elemanlar: ");
10 for (int i = 0; i < sayilar.length; i++) {
11     System.out.print(sayilar[i] + " ");
12 }
```

DEĞİŞKENLER

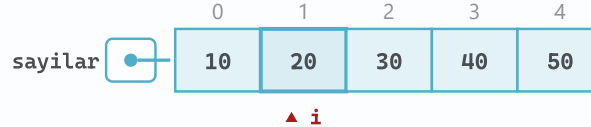
`sayilar = [10, 20, 30, 40, 50]`

`i = 0`

ÇIKTI

```
birinci eleman: 10
ucuncu eleman: 30
Elemanlar: 10
```

DiziOrnegi: Kodu Adım Adım İzle



i = 1 : 1 < 5 doğru → sayilar[1] = 20 yazılır.

Adım 11 / 15

```
1 int[] sayilar = new int[5];
2 sayilar[0] = 10;
3 sayilar[1] = 20;
4 sayilar[2] = 30;
5 sayilar[3] = 40;
6 sayilar[4] = 50;
7 System.out.println("birinci eleman: " + sayilar[0]);
8 System.out.println("ucuncu eleman: " + sayilar[2]);
9 System.out.print("Elemanlar: ");
10 for (int i = 0; i < sayilar.length; i++) {
11     System.out.print(sayilar[i] + " ");
12 }
```

DEĞİŞKENLER

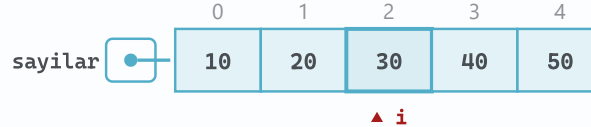
sayilar = [10, 20, 30, 40, 50]

i = 1

ÇIKTI

```
birinci eleman: 10
ucuncu eleman: 30
Elemanlar: 10 20
```

DiziOrnegi: Kodu Adım Adım İzle



`i = 2` : `2 < 5` doğru → `sayilar[2]` = 30 yazılır.

Adım 12 / 15

```
1 int[] sayilar = new int[5];
2 sayilar[0] = 10;
3 sayilar[1] = 20;
4 sayilar[2] = 30;
5 sayilar[3] = 40;
6 sayilar[4] = 50;
7 System.out.println("birinci eleman: " + sayilar[0]);
8 System.out.println("ucuncu eleman: " + sayilar[2]);
9 System.out.print("Elemanlar: ");
10 for (int i = 0; i < sayilar.length; i++) {
11     System.out.print(sayilar[i] + " ");
12 }
```

DEĞİŞKENLER

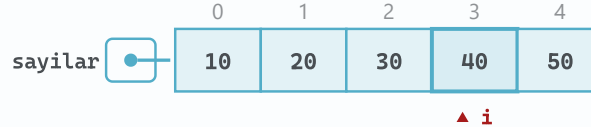
`sayilar = [10, 20, 30, 40, 50]`

`i = 2`

ÇIKTI

```
birinci eleman: 10
ucuncu eleman: 30
Elemanlar: 10 20 30
```

DiziOrnegi: Kodu Adım Adım İzle



`i = 3` : `3 < 5` doğru → `sayilar[3]` = 40 yazılır.

Adım 13 / 15

```
1 int[] sayilar = new int[5];
2 sayilar[0] = 10;
3 sayilar[1] = 20;
4 sayilar[2] = 30;
5 sayilar[3] = 40;
6 sayilar[4] = 50;
7 System.out.println("birinci eleman: " + sayilar[0]);
8 System.out.println("ucuncu eleman: " + sayilar[2]);
9 System.out.print("Elemanlar: ");
10 for (int i = 0; i < sayilar.length; i++) {
11     System.out.print(sayilar[i] + " ");
12 }
```

DEĞİŞKENLER

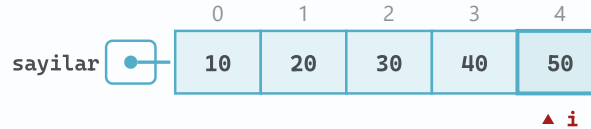
`sayilar = [10, 20, 30, 40, 50]`

`i = 3`

ÇIKTI

```
birinci eleman: 10
ucuncu eleman: 30
Elemanlar: 10 20 30 40
```

DiziOrnegi: Kodu Adım Adım İzle



i = 4 : 4 < 5 doğru → sayilar[4] = 50 yazılır.

Adım 14 / 15

```
1 int[] sayilar = new int[5];
2 sayilar[0] = 10;
3 sayilar[1] = 20;
4 sayilar[2] = 30;
5 sayilar[3] = 40;
6 sayilar[4] = 50;
7 System.out.println("birinci eleman: " + sayilar[0]);
8 System.out.println("ucuncu eleman: " + sayilar[2]);
9 System.out.print("Elemanlar: ");
10 for (int i = 0; i < sayilar.length; i++) {
11     System.out.print(sayilar[i] + " ");
12 }
```

DEĞİŞKENLER

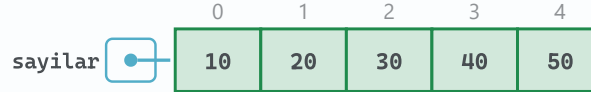
sayilar = [10, 20, 30, 40, 50]

i = 4

ÇIKTI

```
birinci eleman: 10
ucuncu eleman: 30
Elemanlar: 10 20 30 40 50
```

DiziOrnegi: Kodu Adım Adım İzle



`i = 5` : `5 < sayilar.length` yanlış → döngü biter. Son geçerli indis **`length - 1 = 4`**tür.

Adım 15 / 15

```
1 int[] sayilar = new int[5];
2 sayilar[0] = 10;
3 sayilar[1] = 20;
4 sayilar[2] = 30;
5 sayilar[3] = 40;
6 sayilar[4] = 50;
7 System.out.println("birinci eleman: " + sayilar[0]);
8 System.out.println("ucuncu eleman: " + sayilar[2]);
9 System.out.print("Elemanlar: ");
10 ▶ for (int i = 0; i < sayilar.length; i++) {
11     System.out.print(sayilar[i] + " ");
12 }
```

DEĞİŞKENLER

`sayilar = [10, 20, 30, 40, 50]`

`i = 5`

ÇIKTI

```
birinci eleman: 10
ucuncu eleman: 30
Elemanlar: 10 20 30 40 50
```

for-each Döngüsü

```
int[] notlar = {70, 85, 90};  
// Klasik for: indis elimizde  
for (int i = 0; i < notlar.length; i++) {  
    System.out.println(i + ". not: " + notlar[i]);  
}  
// for-each: yalnızca değerler  
int toplam = 0;  
for (int not : notlar) {  
    toplam += not;  
}  
System.out.println("Toplam: " + toplam);
```

ÇIKTI

```
0. not: 70  
1. not: 85  
2. not: 90  
Toplam: 245
```

- Gelişmiş for (enhanced for / for-each): `for (tür eleman : dizi)` her turda sıradaki öğeyi `eleman` değişkenine koyar.
- İndis yazılmaz; bu yüzden sınır dışı hatası yapılamaz.
- Yalnızca **okumak** içindir: `not = 0;` yazmak dizideki öğeyi değiştirmez, yalnızca kopyayı değiştirir.
- İndise ihtiyaç varsa (yazmak, konum bildirmek, iki komşuyu karşılaştırmak) klasik `for` kullanılır.

for-each: Değer Gelir, Hücre Gelmez

```

1 ► int[] puanlar = {60, 75, 95};
2 int toplam = 0;
3 for (int p : puanlar) {
4     toplam += p;
5 }
6 System.out.println("Toplam: " + toplam);
7 for (int p : puanlar) {
8     p = 0;
9 }
10 System.out.println(puanlar[0] + " " +
    puanlar[2]);
    
```

	0	1	2
puanlar	60	75	95

Heap'te üç hücreli bir dizi var; `puanlar` onun referansını tutar.

Adım 1 / 12

DEĞİŞKENLER

`puanlar = [60, 75, 95]`

ÇIKTI



for-each: Değer Gelir, Hücre Gelmez

```
1 int[] puanlar = {60, 75, 95};  
2  
3 ► int toplam = 0;  
4 for (int p : puanlar) {  
5     toplam += p;  
6 }  
7 System.out.println("Toplam: " + toplam);  
8 for (int p : puanlar) {  
9     p = 0;  
10 }  
11 System.out.println(puanlar[0] + " " +  
12     puanlar[2]);
```

	0	1	2
puanlar	60	75	95

toplam 0

toplam 0 ile başlar.

Adım 2 / 12

DEĞİŞKENLER

puanlar = [60, 75, 95]

toplam = 0

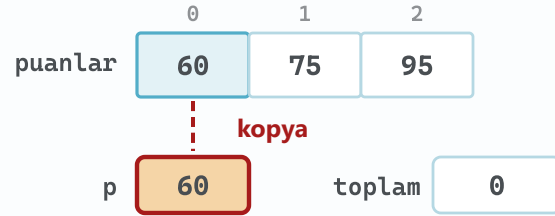
ÇIKTI



for-each: Değer Gelir, Hücre Gelmez

```

1 int[] puanlar = {60, 75, 95};
2 int toplam = 0;
▶ 3 for (int p : puanlar) {
4     toplam += p;
5 }
6 System.out.println("Toplam: " + toplam);
7 for (int p : puanlar) {
8     p = 0;
9 }
10 System.out.println(puanlar[0] + " " +
    puanlar[2]);
    
```



İlk tur: döngü `puanlar[0]` 'ın **kopyasını** `p` 'ye koyar, `p` = 60. İndis yazılmaz.

Adım 3 / 12

DEĞİŞKENLER

`puanlar = [60, 75, 95]`

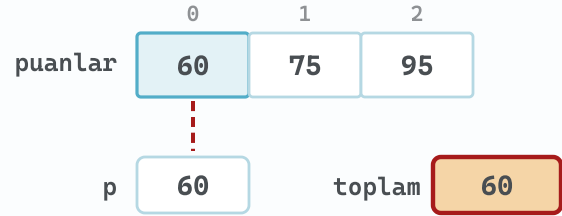
`toplam = 0`

`p = 60`

ÇIKTI

for-each: Değer Gelir, Hücre Gelmez

```
1 int[] puanlar = {60, 75, 95};
2 int toplam = 0;
3 for (int p : puanlar) {
4     toplam += p;
5 }
6 System.out.println("Toplam: " + toplam);
7 for (int p : puanlar) {
8     p = 0;
9 }
10 System.out.println(puanlar[0] + " " + puanlar[2]);
```



toplam += p → 60.

Adım 4 / 12

DEĞİŞKENLER

puanlar = [60, 75, 95]

toplam = 60

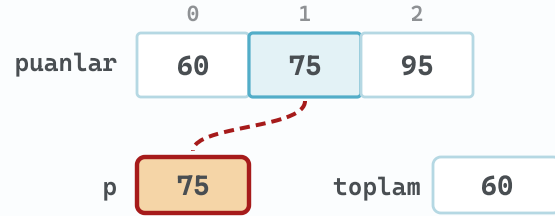
p = 60

ÇIKTI

for-each: Değer Gelir, Hücre Gelmez

```

1 int[] puanlar = {60, 75, 95};
2 int toplam = 0;
▶ 3 for (int p : puanlar) {
4     toplam += p;
5 }
6 System.out.println("Toplam: " + toplam);
7 for (int p : puanlar) {
8     p = 0;
9 }
10 System.out.println(puanlar[0] + " " +
    puanlar[2]);
    
```



İkinci tur: $p = 75$ (sıradaki öğeyi döngü kendisi verir).

Adım 5 / 12

DEĞİŞKENLER

puanlar = [60, 75, 95]

toplam = 60

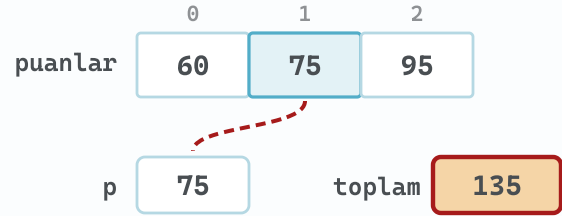
p = 75

ÇIKTI

for-each: Değer Gelir, Hücre Gelmez

```

1 int[] puanlar = {60, 75, 95};
2 int toplam = 0;
3 for (int p : puanlar) {
4     toplam += p;
5 }
6 System.out.println("Toplam: " + toplam);
7 for (int p : puanlar) {
8     p = 0;
9 }
10 System.out.println(puanlar[0] + " " +
    puanlar[2]);
    
```



$\text{toplam} = 60 + 75 = 135.$

Adım 6 / 12

DEĞİŞKENLER

puanlar = [60, 75, 95]

toplam = 135

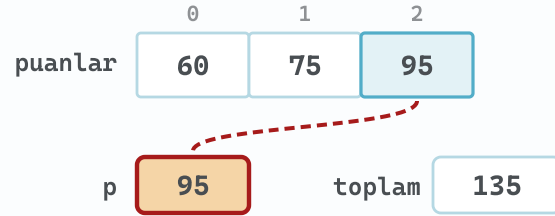
p = 75

ÇIKTI

for-each: Değer Gelir, Hücre Gelmez

```

1 int[] puanlar = {60, 75, 95};
2 int toplam = 0;
▶ 3 for (int p : puanlar) {
4     toplam += p;
5 }
6 System.out.println("Toplam: " + toplam);
7 for (int p : puanlar) {
8     p = 0;
9 }
10 System.out.println(puanlar[0] + " " +
    puanlar[2]);
    
```



Son tur: p = 95.

Adım 7 / 12

DEĞİŞKENLER

puanlar = [60, 75, 95]

toplam = 135

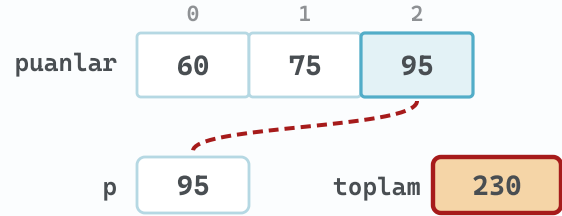
p = 95

ÇIKTI

for-each: Değer Gelir, Hücre Gelmez

```

1 int[] puanlar = {60, 75, 95};
2 int toplam = 0;
3 for (int p : puanlar) {
4     toplam += p;
5 }
6 System.out.println("Toplam: " + toplam);
7 for (int p : puanlar) {
8     p = 0;
9 }
10 System.out.println(puanlar[0] + " " +
    puanlar[2]);
    
```



toplam = 230. Öğe kalmadı, döngü biter.

Adım 8 / 12

DEĞİŞKENLER

puanlar = [60, 75, 95]

toplam = 230

p = 95

ÇIKTI

for-each: Değer Gelir, Hücre Gelmez

```

1 int[] puanlar = {60, 75, 95};
2 int toplam = 0;
3 for (int p : puanlar) {
4     toplam += p;
5 }
6 System.out.println("Toplam: " + toplam);
7 for (int p : puanlar) {
8     p = 0;
9 }
10 System.out.println(puanlar[0] + " " +
    puanlar[2]);
    
```

	0	1	2
puanlar	60	75	95

toplam 230

Toplam: 230 yazılır.

Adım 9 / 12

DEĞİŞKENLER

puanlar = [60, 75, 95]

toplam = 230

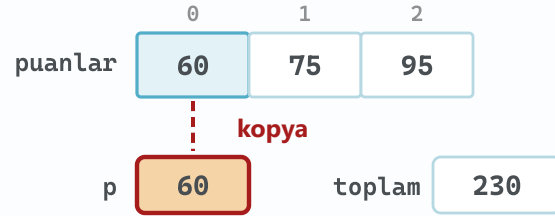
ÇIKTI

Toplam: 230

for-each: Değer Gelir, Hücre Gelmez

```

1 int[] puanlar = {60, 75, 95};
2 int toplam = 0;
3 for (int p : puanlar) {
4     toplam += p;
5 }
6 System.out.println("Toplam: " + toplam);
▶ for (int p : puanlar) {
8     p = 0;
9 }
10 System.out.println(puanlar[0] + " " +
    puanlar[2]);
    
```



İkinci döngü: p yine puanlar[0] 'ın kopyası, 60.

Adım 10 / 12

DEĞİŞKENLER

puanlar = [60, 75, 95]

toplam = 230

p = 60

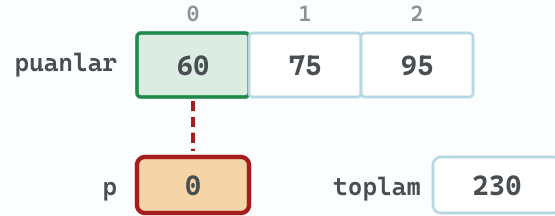
ÇIKTI

Toplam: 230

for-each: Değer Gelir, Hücre Gelmez

```

1 int[] puanlar = {60, 75, 95};
2 int toplam = 0;
3 for (int p : puanlar) {
4     toplam += p;
5 }
6 System.out.println("Toplam: " + toplam);
7 for (int p : puanlar) {
8     p = 0;
9 }
10 System.out.println(puanlar[0] + " " +
    puanlar[2]);
    
```



`p = 0` yalnızca **kopyayı** değiştirir; dizideki 60 yerinde durur. Kalan turlar da aynıdır.

Adım 11 / 12

DEĞİŞKENLER

`puanlar = [60, 75, 95]`

`toplam = 230`

`p = 0`

ÇIKTI

Toplam: 230

for-each: Değer Gelir, Hücre Gelmez

```
1 int[] puanlar = {60, 75, 95};
2 int toplam = 0;
3 for (int p : puanlar) {
4     toplam += p;
5 }
6 System.out.println("Toplam: " + toplam);
7 for (int p : puanlar) {
8     p = 0;
9 }
▶ System.out.println(puanlar[0] + " " +
    puanlar[2]);
```

	0	1	2
puanlar	60	75	95

toplam 230

60 95: dizi hiç değişmedi. for-each ile diziye **yazılamaz**; yazmak için klasik for ve `puanlar[i]` gerekir.

Adım 12 / 12

DEĞİŞKENLER

puanlar = [60, 75, 95]

toplam = 230

ÇIKTI

```
Toplam: 230
60 95
```

Dizi Değişkeni Bir Referanstır

```
► int[] a = {1, 2, 3};  
2 int[] b = a;           // kopya değil!  
3 b[0] = 99;  
4 System.out.println(a[0]);  
5 int[] c = a.clone(); // gerçek kopya  
6 c[1] = 77;  
7 System.out.println(a[1] + " " + c[1]);  
8 System.out.println(a == b);  
9 System.out.println(a == c);
```

Yığın (stack)



Heap

dizi #1



Heap'te bir dizi oluşur; **a** değişkeninde dizinin kendisi değil, **adres (referansı)** vardır.

Adım 1 / 8

ÇIKTI



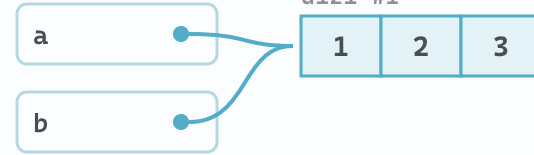
Dizi Değişkeni Bir Referanstır

```

1 int[] a = {1, 2, 3};
▶ int[] b = a;           // kopya değil!
3 b[0] = 99;
4 System.out.println(a[0]);
5 int[] c = a.clone(); // gerçek kopya
6 c[1] = 77;
7 System.out.println(a[1] + " " + c[1]);
8 System.out.println(a == b);
9 System.out.println(a == c);
    
```

Yığın (stack)

Heap



`b = a` yalnızca **adresli kopyalar**. Artık iki değişken **aynı** diziyi gösterir (takma ad, alias).

Adım 2 / 8

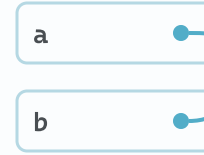
ÇIKTI



Dizi Değişkeni Bir Referanstır

```
1 int[] a = {1, 2, 3};
2 int[] b = a;           // kopya değil!
▶ b[0] = 99;
4 System.out.println(a[0]);
5 int[] c = a.clone(); // gerçek kopya
6 c[1] = 77;
7 System.out.println(a[1] + " " + c[1]);
8 System.out.println(a == b);
9 System.out.println(a == c);
```

Yığın (stack)



Heap

dizi #1



b[0] = 99 ortak diziye değiştirir...

Adım 3 / 8

ÇIKTI

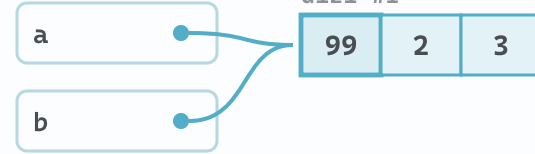


Dizi Değişkeni Bir Referanstır

```
1 int[] a = {1, 2, 3};
2 int[] b = a;           // kopya değil!
3 b[0] = 99;
4 ► System.out.println(a[0]);
5 int[] c = a.clone(); // gerçek kopya
6 c[1] = 77;
7 System.out.println(a[1] + " " + c[1]);
8 System.out.println(a == b);
9 System.out.println(a == c);
```

Yığın (stack)

Heap



...bu yüzden `a[0]` da **99** olur.

Adım 4 / 8

ÇIKTI

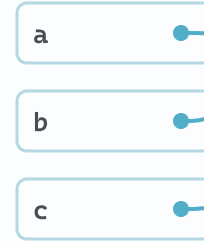
99

Dizi Değişkeni Bir Referanstır

```

1 int[] a = {1, 2, 3};
2 int[] b = a;           // kopya değil!
3 b[0] = 99;
4 System.out.println(a[0]);
5 ▶ int[] c = a.clone(); // gerçek kopya
6 c[1] = 77;
7 System.out.println(a[1] + " " + c[1]);
8 System.out.println(a == b);
9 System.out.println(a == c);
    
```

Yığın (stack)



Heap

dizi #1



dizi #2 (kopya)



`clone()` heap'te **yeni** bir dizi oluşturup öğeleri kopyalar; `c` bu yeni diziyi gösterir.

Adım 5 / 8

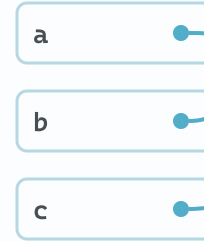
ÇIKTI

99

Dizi Değişkeni Bir Referanstır

```
1 int[] a = {1, 2, 3};
2 int[] b = a;           // kopya değil!
3 b[0] = 99;
4 System.out.println(a[0]);
5 int[] c = a.clone(); // gerçek kopya
6 ▶ c[1] = 77;
7 System.out.println(a[1] + " " + c[1]);
8 System.out.println(a == b);
9 System.out.println(a == c);
```

Yığın (stack)



Heap

dizi #1



dizi #2 (kopya)



`c[1] = 77` yalnızca kopyayı değiştirir; asıl dizi etkilenmez.

Adım 6 / 8

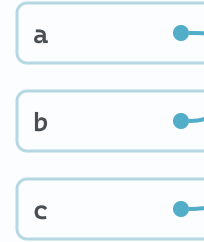
ÇIKTI

99

Dizi Değişkeni Bir Referanstır

```
1 int[] a = {1, 2, 3};
2 int[] b = a;           // kopya değil!
3 b[0] = 99;
4 System.out.println(a[0]);
5 int[] c = a.clone(); // gerçek kopya
6 c[1] = 77;
7 ▶ System.out.println(a[1] + " " + c[1]);
8 System.out.println(a == b);
9 System.out.println(a == c);
```

Yığın (stack)

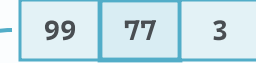


Heap

dizi #1



dizi #2 (kopya)



a[1] hâlâ 2, c[1] ise 77.

Adım 7 / 8

ÇIKTI

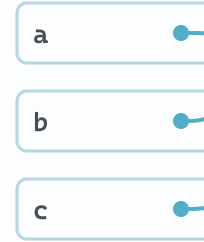
```
99
2 77
```

Dizi Değişkeni Bir Referanstır

```

1 int[] a = {1, 2, 3};
2 int[] b = a;           // kopya değil!
3 b[0] = 99;
4 System.out.println(a[0]);
5 int[] c = a.clone(); // gerçek kopya
6 c[1] = 77;
7 System.out.println(a[1] + " " + c[1]);
▶ System.out.println(a == b);
▶ System.out.println(a == c);
    
```

Yığın (stack)



Heap



`==` referansları (adresleri) karşılaştırır: `a == b` doğru (aynı dizi), `a == c` yanlış (farklı diziler).

Adım 8 / 8

ÇIKTI

```

99
2 77
true
false
    
```

JAVA'DA DİZİLER

java.util.Arrays Sınıfı

```
int[] dizi = {5, 3, 8, 1};
System.out.println(Arrays.toString(dizi));
Arrays.sort(dizi);
System.out.println(Arrays.toString(dizi));
int[] kopya = Arrays.copyOf(dizi, 6);
System.out.println(Arrays.toString(kopya));
int[] dolu = new int[4];
Arrays.fill(dolu, 7);
System.out.println(Arrays.toString(dolu));
int[] x = {1, 2};
int[] y = {1, 2};
System.out.println((x == y) + " " + Arrays.equals(x, y));
System.out.println(Arrays.binarySearch(dizi, 5));
```

ÇIKTI

```
[5, 3, 8, 1]
[1, 3, 5, 8]
[1, 3, 5, 8, 0, 0]
[7, 7, 7, 7]
false true
2
```

Metot (import java.util.Arrays;)	Görevi
<code>Arrays.toString(d)</code>	Diziyi metne çevirir; <code>println(d)</code> ise <code>[I@7ef20235]</code> gibi bir kimlik yazar
<code>Arrays.sort(d)</code> · <code>binarySearch</code>	Yerinde sıralar · sıralı dizide indis arar (5 → 2)
<code>Arrays.copyOf(d, n)</code> · <code>d.clone()</code>	Yeni dizi oluşturup kopyalar; fazla hücreler 0 olur
<code>Arrays.fill</code> · <code>Arrays.equals</code>	Tüm hücrelere aynı değeri yazar · içerikleri karşılaştırır (<code>==</code> adres karşılaştırır)

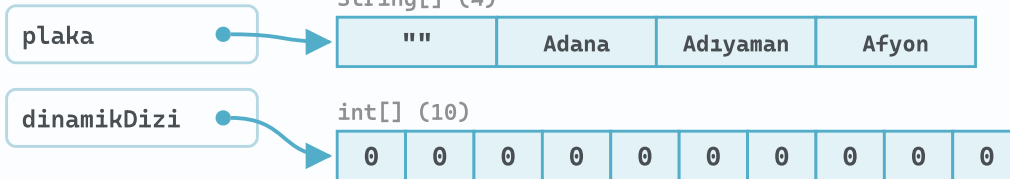
Diziler Bellekte Nerede?

```
String[] plaka = {"", "Adana", "Adıyaman", "Afyon"};
System.out.println(plaka[1]);
int[] dinamikDizi = new int[10]; // heap'te yer ayrılır
System.out.println(dinamikDizi.length);
dinamikDizi = null; // referans koparıldı
System.out.println(dinamikDizi == null);
```

- **C/C++'ta** `int a[10];` yığında durur; `malloc` / `new` ile heap'te ayrılan dizi **elle** serbest bırakılır.
- Çıktı: `Adana` , `10` , `true` .
- **Java'da tüm diziler heap'tedir**; boyut çalışma zamanında verilebilir, sonra sabittir. Serbest bırakma **otomatiktir**.

Yığın (stack)

Heap



Her iki dizi de **heap'te** oluşur; yığında (stack) yalnızca referans değişkenleri durur. (`String[]` hücreleri de aslında ayrı `String` nesnelere referanstır.)

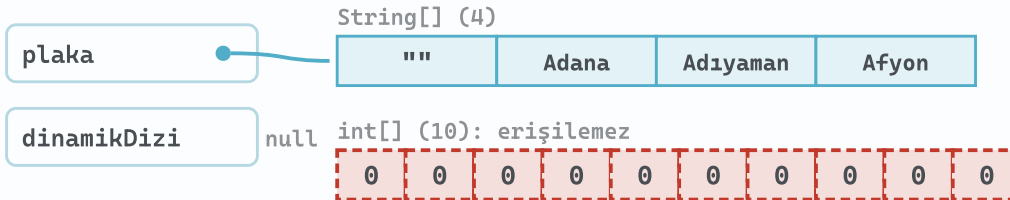
Diziler Bellekte Nerede?

```
String[] plaka = {"", "Adana", "Adıyaman", "Afyon"};
System.out.println(plaka[1]);
int[] dinamikDizi = new int[10]; // heap'te yer ayrılır
System.out.println(dinamikDizi.length);
dinamikDizi = null; // referans koparıldı
System.out.println(dinamikDizi == null);
```

- **C/C++'ta** `int a[10];` yığında durur; `malloc` / `new` ile heap'te ayrılan dizi **elle** serbest bırakılır.
- Çıktı: `Adana` , `10` , `true` .
- **Java'da tüm diziler heap'tedir**; boyut çalışma zamanında verilebilir, sonra sabittir. Serbest bırakma **otomatiktir**.

Yığın (stack)

Heap



`dinamikDizi = null` **diziye silmez**, yalnızca bağı koparır. Dizi artık erişilemez olduğundan **çöp toplayıcı (garbage collector)** uygun bir anda belleği geri alır.

Diziler Bellekte Nerede?

```
String[] plaka = {"", "Adana", "Adıyaman", "Afyon"};
System.out.println(plaka[1]);
int[] dinamikDizi = new int[10]; // heap'te yer ayrılır
System.out.println(dinamikDizi.length);
dinamikDizi = null; // referans koparıldı
System.out.println(dinamikDizi == null);
```

- **C/C++**'ta `int a[10];` yığında durur; `malloc` / `new` ile heap'te ayrılan dizi **elle** serbest bırakılır.
- Çıktı: `Adana` , `10` , `true` .
- **Java**'da **tüm diziler heap'tedir**; boyut çalışma zamanında verilebilir, sonra sabittir. Serbest bırakma **otomatiktir**.

Yığın (stack)

Heap

String[] (4)

""	Adana	Adıyaman	Afyon
----	-------	----------	-------

Metot bitince yığın çerçevesi ve içindeki referanslar **otomatik** kalkar. Başka referans kalmayan diziler de zamanla toplanır. Java'da `free` / `delete` yoktur.

İndis Sınır Dışı Hatası

- Dizinin sınırları (0 ... length – 1) dışındaki bir indise erişildiğinde Java çalışma zamanında `ArrayIndexOutOfBoundsException` fırlatır (`IndexOutOfBoundsException` 'ın alt sınıfı).
- Yakalanmazsa program o satırda **çöker**; C'deki gibi sessizce başka bir belleği bozmaz.

```
int[] dizi = {10, 20, 30, 40, 50};
int index = 5;
// Dizi uzunluğunu kontrol etme
if (index >= 0 && index < dizi.length) {
    int deger = dizi[index];
    System.out.println("Değer: " + deger);
} else {
    System.out.println("Geçersiz indis: " + index);
}
// Kontrolsüz erişim:
System.out.println(dizi[index]);
```

ÇIKTI

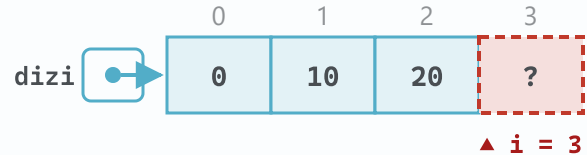
```
Geçersiz indis: 5
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException:
Index 5 out of bounds for length 5
    at
    SinirDisi.main(SinirDisi.java:13)
```

Mesaj; hatalı indisi (5), dizinin uzunluğunu (5) ve satır numarasını söyler: hatayı bulmak için ilk bakılacak yer.

Klasik Hata: `<=` ile Döngü

```
int[] dizi = new int[3];  
for (int i = 0; i <= dizi.length; i++) {  
    dizi[i] = i * 10;  
}
```

```
Exception in thread "main"  
java.lang.ArrayIndexOutOfBoundsException: Index  
3 out of bounds for length 3  
    at SinirDongu.main(SinirDongu.java:5)
```



- `i = 3` olduğunda koşul `3 <= 3` doğru kalır ve `dizi[3]` yazılmaya çalışılır.
- Kural: `i < dizi.length` yazın; sabit sayı (`i < 3`) yerine `length` kullanın ki boyut değişince kod bozulmasın.

<= ile Döngü: Hata Hangi Turda?

```
► int[] kare = new int[4];  
2 for (int i = 0; i <= kare.length; i++) {  
3     kare[i] = i * i;  
4 }  
5 System.out.println("bitti");
```



4 hücreli dizi; geçerli indisler 0 ... 3, `kare.length` = 4.

Adım 1 / 11

DEĞİŞKENLER

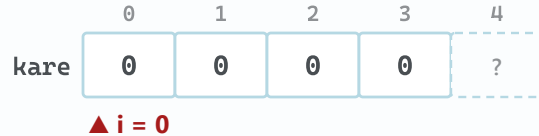
`kare = [0, 0, 0, 0]`

ÇIKTI



<= ile Döngü: Hata Hangi Turda?

```
1 int[] kare = new int[4];  
▶ for (int i = 0; i <= kare.length; i++) {  
3     kare[i] = i * i;  
4 }  
5 System.out.println("bitti");
```



i = 0: koşul `0 <= 4` doğru.

Adım 2 / 11

DEĞİŞKENLER

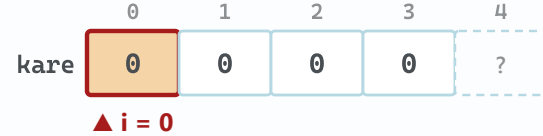
kare = [0, 0, 0, 0]

i = 0

ÇIKTI

<= ile Döngü: Hata Hangi Turda?

```
1 int[] kare = new int[4];  
2 for (int i = 0; i <= kare.length; i++) {  
    ▶ kare[i] = i * i;  
4 }  
5 System.out.println("bitti");
```



kare[0] = 0 .

Adım 3 / 11

DEĞİŞKENLER

kare = [0, 0, 0, 0]

i = 0

ÇIKTI

<= ile Döngü: Hata Hangi Turda?

```
1 int[] kare = new int[4];  
▶ for (int i = 0; i <= kare.length; i++) {  
3     kare[i] = i * i;  
4 }  
5 System.out.println("bitti");
```



`i = 1`: `1 <= 4` doğru.

Adım 4 / 11

DEĞİŞKENLER

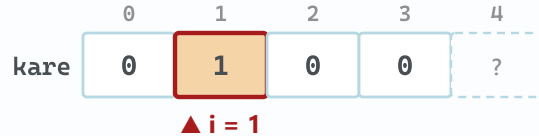
`kare = [0, 0, 0, 0]`

`i = 1`

ÇIKTI

<= ile Döngü: Hata Hangi Turda?

```
1 int[] kare = new int[4];  
2 for (int i = 0; i <= kare.length; i++) {  
3     kare[i] = i * i;  
4 }  
5 System.out.println("bitti");
```



`kare[1] = 1` .

Adım 5 / 11

DEĞİŞKENLER

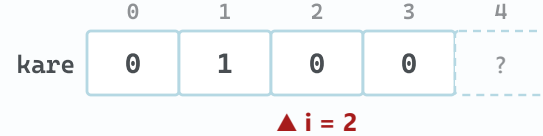
`kare = [0, 1, 0, 0]`

`i = 1`

ÇIKTI

<= ile Döngü: Hata Hangi Turda?

```
1 int[] kare = new int[4];  
▶ for (int i = 0; i <= kare.length; i++) {  
3     kare[i] = i * i;  
4 }  
5 System.out.println("bitti");
```



i = 2: doğru.

Adım 6 / 11

DEĞİŞKENLER

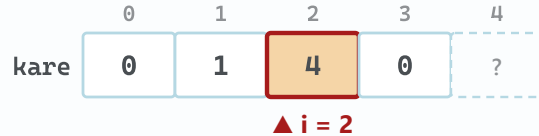
kare = [0, 1, 0, 0]

i = 2

ÇIKTI

<= ile Döngü: Hata Hangi Turda?

```
1 int[] kare = new int[4];  
2 for (int i = 0; i <= kare.length; i++) {  
3     kare[i] = i * i;  
4 }  
5 System.out.println("bitti");
```



`kare[2] = 4` .

Adım 7 / 11

DEĞİŞKENLER

`kare = [0, 1, 4, 0]`

`i = 2`

ÇIKTI

<= ile Döngü: Hata Hangi Turda?

```
1 int[] kare = new int[4];  
▶ for (int i = 0; i <= kare.length; i++) {  
3     kare[i] = i * i;  
4 }  
5 System.out.println("bitti");
```



i = 3: doğru.

Adım 8 / 11

DEĞİŞKENLER

kare = [0, 1, 4, 0]

i = 3

ÇIKTI



<= ile Döngü: Hata Hangi Turda?

```
1 int[] kare = new int[4];
2 for (int i = 0; i <= kare.length; i++) {
3     kare[i] = i * i;
4 }
5 System.out.println("bitti");
```



`kare[3] = 9` : son geçerli hücre doldu.

Adım 9 / 11

DEĞİŞKENLER

`kare = [0, 1, 4, 9]`

`i = 3`

ÇIKTI

<= ile Döngü: Hata Hangi Turda?

```
1 int[] kare = new int[4];  
▶ for (int i = 0; i <= kare.length; i++) {  
3     kare[i] = i * i;  
4 }  
5 System.out.println("bitti");
```



i = 4: 4 <= 4 de **doğru!** < yazılsaydı döngü burada biterdi.

Adım 10 / 11

DEĞİŞKENLER

kare = [0, 1, 4, 9]

i = 4

ÇIKTI



<= ile Döngü: Hata Hangi Turda?

```
1 int[] kare = new int[4];  
2 for (int i = 0; i <= kare.length; i++) {  
3     kare[i] = i * i;  
4 }  
5 System.out.println("bitti");
```



ArrayIndexOutOfBoundsException

kare[4] yok: JVM

ArrayIndexOutOfBoundsException fırlatır, program bu satırda durur; "bitti" hiç yazılmaz.

Adım 11 / 11

DEĞİŞKENLER

kare = [0, 1, 4, 9]

ÇIKTI

```
Exception in thread "main"  
java.lang.ArrayIndexOutOfBoundsException:  
Index 4 out of bounds for length 4
```

Kullanıcıdan Dizi Okuma

```
import java.util.Scanner;
public class OrtalamaHesaplama {
    public static void main(String[] args) {
        // Kullanıcıdan dizi boyutunu al
        Scanner scanner = new Scanner(System.in);
        System.out.print("boyutu girin: ");
        int boyut = scanner.nextInt();
        // Kullanıcıdan elemanları al ve dizi oluştur
        double[] dizi = new double[boyut];
        for (int i = 0; i < boyut; i++) {
            System.out.print((i + 1) + ". elemani girin: ");
            dizi[i] = scanner.nextDouble();
        }
        // ... diziyle hesaplamalar
    }
}
```

ÖRNEK ÇALIŞMA (GİRDİLER KALIN)

```
boyutu girin: 4
1. elemani girin: 7
2. elemani girin: 3
3. elemani girin: 9
4. elemani girin: 5
```

- Dizi boyutu **kullanıcıdan** gelir: `new double[boyut]`. Boyut çalışma zamanında belirlenir, sonra değişmez.
- İstemde `i + 1` yazılır (kullanıcı 1'den sayar), diziye `dizi[i]` ile yazılır (0'dan).
- Ortalama, ortanca ve mod programları bu okuma bölümüyle başlar.

OrtalamaHesaplama: Kodu Adım Adım İzle

```
1 Scanner scanner = new Scanner(System.in);
2 System.out.print("boyutu girin: ");
3 ▶ int boyut = scanner.nextInt();
4 double[] dizi = new double[boyut];
5 for (int i = 0; i < boyut; i++) {
6     System.out.print((i + 1) + ". elemani girin: ");
7     dizi[i] = scanner.nextDouble();
8 }
9 double toplam = 0;
10 for (int i = 0; i < boyut; i++) {
11     toplam += dizi[i];
12 }
13 double ortalama = toplam / boyut;
14 System.out.print("Elemanlar: ");
15 for (int i = 0; i < boyut; i++) {
16     System.out.print(dizi[i] + " ");
17 }
18 System.out.println();
19 System.out.println("Ortalama: " + ortalama);
```

Kullanıcı boyut olarak **4** girer.

`nextInt()` klavyeden bir tam sayı okur: `boyut`
= **4**. Dizinin boyutu ancak şimdi, **çalışma**
zamanında belli oldu.

Adım 1 / 13

DEĞİŞKENLER

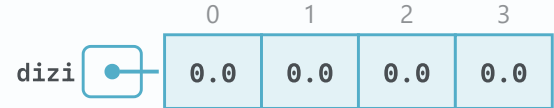
`boyut = 4`

ÇIKTI

boyutu girin: 4

OrtalamaHesaplama: Kodu Adım Adım İzle

```
1 Scanner scanner = new Scanner(System.in);
2 System.out.print("boyutu girin: ");
3 int boyut = scanner.nextInt();
4 double[] dizi = new double[boyut];
5 for (int i = 0; i < boyut; i++) {
6     System.out.print((i + 1) + ". elemani girin: ");
7     dizi[i] = scanner.nextDouble();
8 }
9 double toplam = 0;
10 for (int i = 0; i < boyut; i++) {
11     toplam += dizi[i];
12 }
13 double ortalama = toplam / boyut;
14 System.out.print("Elemanlar: ");
15 for (int i = 0; i < boyut; i++) {
16     System.out.print(dizi[i] + " ");
17 }
18 System.out.println();
19 System.out.println("Ortalama: " + ortalama);
```



new double[boyut] : 4 hücreli dizi, hepsi 0.0 .

Adım 2 / 13

DEĞİŞKENLER

boyut = 4

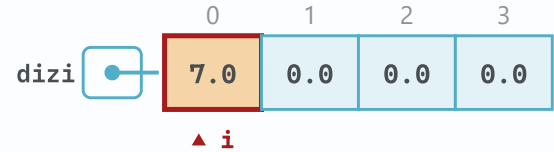
dizi = [0.0, 0.0, 0.0, 0.0]

ÇIKTI

boyutu girin: 4

Ortalama Hesaplama: Kodu Adım Adım İzle

```
1 Scanner scanner = new Scanner(System.in);
2 System.out.print("boyutu girin: ");
3 int boyut = scanner.nextInt();
4 double[] dizi = new double[boyut];
5 for (int i = 0; i < boyut; i++) {
6     System.out.print((i + 1) + ". elemani girin: ");
7     dizi[i] = scanner.nextDouble();
8 }
9 double toplam = 0;
10 for (int i = 0; i < boyut; i++) {
11     toplam += dizi[i];
12 }
13 double ortalama = toplam / boyut;
14 System.out.print("Elemanlar: ");
15 for (int i = 0; i < boyut; i++) {
16     System.out.print(dizi[i] + " ");
17 }
18 System.out.println();
19 System.out.println("Ortalama: " + ortalama);
```



`i = 0` : istem 1. elemani girin: yazılır,
kullanıcı 7 girer → `dizi[0] = 7.0`.

Adım 3 / 13

DEĞİŞKENLER

`boyut = 4`

`i = 0`

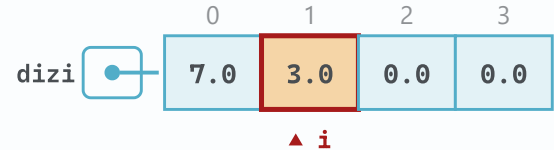
`dizi = [7.0, 0.0, 0.0, 0.0]`

ÇIKTI

```
boyutu girin: 4
1. elemani girin: 7
```

Ortalama Hesaplama: Kodu Adım Adım İzle

```
1 Scanner scanner = new Scanner(System.in);
2 System.out.print("boyutu girin: ");
3 int boyut = scanner.nextInt();
4 double[] dizi = new double[boyut];
5 for (int i = 0; i < boyut; i++) {
6     System.out.print((i + 1) + ". elemanı girin: ");
7     dizi[i] = scanner.nextDouble();
8 }
9 double toplam = 0;
10 for (int i = 0; i < boyut; i++) {
11     toplam += dizi[i];
12 }
13 double ortalama = toplam / boyut;
14 System.out.print("Elemanlar: ");
15 for (int i = 0; i < boyut; i++) {
16     System.out.print(dizi[i] + " ");
17 }
18 System.out.println();
19 System.out.println("Ortalama: " + ortalama);
```



i = 1 : istem 2. elemanı girin: yazılır,
kullanıcı 3 girer → dizi[1] = 3.0 .

Adım 4 / 13

DEĞİŞKENLER

boyut = 4

i = 1

dizi = [7.0, 3.0, 0.0, 0.0]

ÇIKTI

```
boyutu girin: 4
1. elemanı girin: 7
2. elemanı girin: 3
```

Ortalama Hesaplama: Kodu Adım Adım İzle

```
1 Scanner scanner = new Scanner(System.in);
2 System.out.print("boyutu girin: ");
3 int boyut = scanner.nextInt();
4 double[] dizi = new double[boyut];
5 for (int i = 0; i < boyut; i++) {
6     System.out.print((i + 1) + ". elemanı girin: ");
7     dizi[i] = scanner.nextDouble();
8 }
9 double toplam = 0;
10 for (int i = 0; i < boyut; i++) {
11     toplam += dizi[i];
12 }
13 double ortalama = toplam / boyut;
14 System.out.print("Elemanlar: ");
15 for (int i = 0; i < boyut; i++) {
16     System.out.print(dizi[i] + " ");
17 }
18 System.out.println();
19 System.out.println("Ortalama: " + ortalama);
```



i = 2 : istem 3. elemanı girin: yazılır,
kullanıcı 9 girer → dizi[2] = 9.0 .

Adım 5 / 13

DEĞİŞKENLER

boyut = 4

i = 2

dizi = [7.0, 3.0, 9.0, 0.0]

ÇIKTI

```
boyutu girin: 4
1. elemanı girin: 7
2. elemanı girin: 3
3. elemanı girin: 9
```

Ortalama Hesaplama: Kodu Adım Adım İzle

```
1 Scanner scanner = new Scanner(System.in);
2 System.out.print("boyutu girin: ");
3 int boyut = scanner.nextInt();
4 double[] dizi = new double[boyut];
5 for (int i = 0; i < boyut; i++) {
6     System.out.print((i + 1) + ". elemanı girin: ");
7     dizi[i] = scanner.nextDouble();
8 }
9 double toplam = 0;
10 for (int i = 0; i < boyut; i++) {
11     toplam += dizi[i];
12 }
13 double ortalama = toplam / boyut;
14 System.out.print("Elemanlar: ");
15 for (int i = 0; i < boyut; i++) {
16     System.out.print(dizi[i] + " ");
17 }
18 System.out.println();
19 System.out.println("Ortalama: " + ortalama);
```



i = 3 : istem 4. elemanı girin: yazılır,
kullanıcı 5 girer → dizi[3] = 5.0 .

Adım 6 / 13

DEĞİŞKENLER

boyut = 4

i = 3

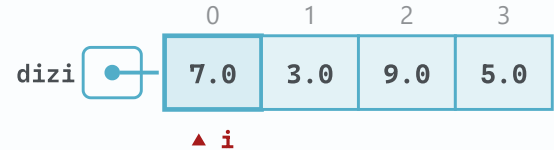
dizi = [7.0, 3.0, 9.0, 5.0]

ÇIKTI

```
boyutu girin: 4
1. elemanı girin: 7
2. elemanı girin: 3
3. elemanı girin: 9
4. elemanı girin: 5
```

Ortalama Hesaplama: Kodu Adım Adım İzle

```
1 Scanner scanner = new Scanner(System.in);
2 System.out.print("boyutu girin: ");
3 int boyut = scanner.nextInt();
4 double[] dizi = new double[boyut];
5 for (int i = 0; i < boyut; i++) {
6     System.out.print((i + 1) + ". elemanı girin: ");
7     dizi[i] = scanner.nextDouble();
8 }
9 double toplam = 0;
10 for (int i = 0; i < boyut; i++) {
11     toplam += dizi[i];
12 }
13 double ortalama = toplam / boyut;
14 System.out.print("Elemanlar: ");
15 for (int i = 0; i < boyut; i++) {
16     System.out.print(dizi[i] + " ");
17 }
18 System.out.println();
19 System.out.println("Ortalama: " + ortalama);
```



`toplam += dizi[0]` → toplam = **7.0**.

Adım 7 / 13

DEĞİŞKENLER

`i = 0`

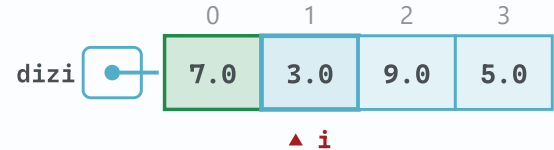
`toplam = 7.0`

ÇIKTI

```
boyutu girin: 4
1. elemanı girin: 7
2. elemanı girin: 3
3. elemanı girin: 9
4. elemanı girin: 5
```

Ortalama Hesaplama: Kodu Adım Adım İzle

```
1 Scanner scanner = new Scanner(System.in);
2 System.out.print("boyutu girin: ");
3 int boyut = scanner.nextInt();
4 double[] dizi = new double[boyut];
5 for (int i = 0; i < boyut; i++) {
6     System.out.print((i + 1) + ". elemanı girin: ");
7     dizi[i] = scanner.nextDouble();
8 }
9 double toplam = 0;
10 for (int i = 0; i < boyut; i++) {
11     toplam += dizi[i];
12 }
13 double ortalama = toplam / boyut;
14 System.out.print("Elemanlar: ");
15 for (int i = 0; i < boyut; i++) {
16     System.out.print(dizi[i] + " ");
17 }
18 System.out.println();
19 System.out.println("Ortalama: " + ortalama);
```



toplam += dizi[1] → toplam = **10.0**.

Adım 8 / 13

DEĞİŞKENLER

i = 1

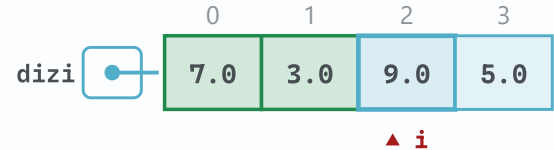
toplam = 10.0

ÇIKTI

```
boyutu girin: 4
1. elemanı girin: 7
2. elemanı girin: 3
3. elemanı girin: 9
4. elemanı girin: 5
```

Ortalama Hesaplama: Kodu Adım Adım İzle

```
1 Scanner scanner = new Scanner(System.in);
2 System.out.print("boyutu girin: ");
3 int boyut = scanner.nextInt();
4 double[] dizi = new double[boyut];
5 for (int i = 0; i < boyut; i++) {
6     System.out.print((i + 1) + ". elemanı girin: ");
7     dizi[i] = scanner.nextDouble();
8 }
9 double toplam = 0;
10 for (int i = 0; i < boyut; i++) {
11     toplam += dizi[i];
12 }
13 double ortalama = toplam / boyut;
14 System.out.print("Elemanlar: ");
15 for (int i = 0; i < boyut; i++) {
16     System.out.print(dizi[i] + " ");
17 }
18 System.out.println();
19 System.out.println("Ortalama: " + ortalama);
```



toplam += dizi[2] → toplam = **19.0**

Adım 9 / 13

DEĞİŞKENLER

i = 2

toplam = 19.0

ÇIKTI

```
boyutu girin: 4
1. elemanı girin: 7
2. elemanı girin: 3
3. elemanı girin: 9
4. elemanı girin: 5
```

Ortalama Hesaplama: Kodu Adım Adım İzle

```
1 Scanner scanner = new Scanner(System.in);
2 System.out.print("boyutu girin: ");
3 int boyut = scanner.nextInt();
4 double[] dizi = new double[boyut];
5 for (int i = 0; i < boyut; i++) {
6     System.out.print((i + 1) + ". elemanı girin: ");
7     dizi[i] = scanner.nextDouble();
8 }
9 double toplam = 0;
10 for (int i = 0; i < boyut; i++) {
11     toplam += dizi[i];
12 }
13 double ortalama = toplam / boyut;
14 System.out.print("Elemanlar: ");
15 for (int i = 0; i < boyut; i++) {
16     System.out.print(dizi[i] + " ");
17 }
18 System.out.println();
19 System.out.println("Ortalama: " + ortalama);
```



toplam += dizi[3] → toplam = **24.0**.

Adım 10 / 13

DEĞİŞKENLER

i = 3

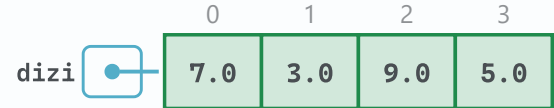
toplam = 24.0

ÇIKTI

```
boyutu girin: 4
1. elemanı girin: 7
2. elemanı girin: 3
3. elemanı girin: 9
4. elemanı girin: 5
```

OrtalamaHesaplama: Kodu Adım Adım İzle

```
1 Scanner scanner = new Scanner(System.in);
2 System.out.print("boyutu girin: ");
3 int boyut = scanner.nextInt();
4 double[] dizi = new double[boyut];
5 for (int i = 0; i < boyut; i++) {
6     System.out.print((i + 1) + ". elemanı girin: ");
7     dizi[i] = scanner.nextDouble();
8 }
9 double toplam = 0;
10 for (int i = 0; i < boyut; i++) {
11     toplam += dizi[i];
12 }
13 ▶ double ortalama = toplam / boyut;
14 System.out.print("Elemanlar: ");
15 for (int i = 0; i < boyut; i++) {
16     System.out.print(dizi[i] + " ");
17 }
18 System.out.println();
19 System.out.println("Ortalama: " + ortalama);
```



ortalama = 24.0 / 4 = 6.0 toplam

double olduğu için bölme ondalıklıdır.

Adım 11 / 13

DEĞİŞKENLER

toplam = 24.0

ortalama = 6.0

ÇIKTI

```
boyutu girin: 4
1. elemanı girin: 7
2. elemanı girin: 3
3. elemanı girin: 9
4. elemanı girin: 5
```

Ortalama Hesaplama: Kodu Adım Adım İzle

```
1 Scanner scanner = new Scanner(System.in);
2 System.out.print("boyutu girin: ");
3 int boyut = scanner.nextInt();
4 double[] dizi = new double[boyut];
5 for (int i = 0; i < boyut; i++) {
6     System.out.print((i + 1) + ". elemanı girin: ");
7     dizi[i] = scanner.nextDouble();
8 }
9 double toplam = 0;
10 for (int i = 0; i < boyut; i++) {
11     toplam += dizi[i];
12 }
13 double ortalama = toplam / boyut;
14 System.out.print("Elemanlar: ");
15 for (int i = 0; i < boyut; i++) {
16     System.out.print(dizi[i] + " ");
17 }
18 ▶ System.out.println();
19 System.out.println("Ortalama: " + ortalama);
```



İkinci döngü öğeleri yan yana yazar; `double` değerler `7.0` biçiminde görünür. Sonra `println()` satırı bitirir.

Adım 12 / 13

DEĞİŞKENLER

toplam = 24.0

ortalama = 6.0

ÇIKTI

```
boyutu girin: 4
1. elemanı girin: 7
2. elemanı girin: 3
3. elemanı girin: 9
4. elemanı girin: 5
Elemanlar: 7.0 3.0 9.0 5.0
```

Ortalama Hesaplama: Kodu Adım Adım İzle

```
1 Scanner scanner = new Scanner(System.in);
2 System.out.print("boyutu girin: ");
3 int boyut = scanner.nextInt();
4 double[] dizi = new double[boyut];
5 for (int i = 0; i < boyut; i++) {
6     System.out.print((i + 1) + ". elemanı girin: ");
7     dizi[i] = scanner.nextDouble();
8 }
9 double toplam = 0;
10 for (int i = 0; i < boyut; i++) {
11     toplam += dizi[i];
12 }
13 double ortalama = toplam / boyut;
14 System.out.print("Elemanlar: ");
15 for (int i = 0; i < boyut; i++) {
16     System.out.print(dizi[i] + " ");
17 }
18 System.out.println();
19 System.out.println("Ortalama: " + ortalama);
```



Sonuç yazdırılır.

Adım 13 / 13

DEĞİŞKENLER

toplam = 24.0

ortalama = 6.0

ÇIKTI

```
boyutu girin: 4
1. elemanı girin: 7
2. elemanı girin: 3
3. elemanı girin: 9
4. elemanı girin: 5
Elemanlar: 7.0 3.0 9.0 5.0
Ortalama: 6.0
```

OrtancaDegerHesaplama

```
// boyut ve dizi, OrtalamaHesaplama'daki gibi okunur
Arrays.sort(dizi); // Diziyi sırala
double ortanca;
if (boyut % 2 == 0) { // Çift sayıda eleman
    int orta1 = boyut / 2 - 1;
    int orta2 = boyut / 2;
    ortanca = (dizi[orta1] + dizi[orta2]) / 2;
} else { // Tek sayıda eleman
    ortanca = dizi[boyut / 2];
}
System.out.println("Ortanca Değer: " + ortanca);
```

ÖRNEK ÇALIŞMA

girilenler: 7 3 9 5 1

Elemanlar (sıralı): 1.0 3.0 5.0 7.0 9.0

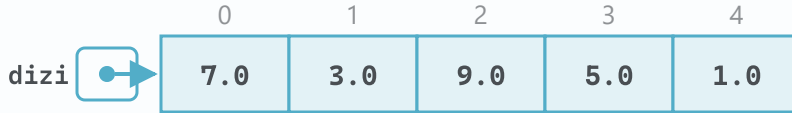
Ortanca Değer: 5.0

Ortanca (medyan)

Sıralı dizinin **ortasındaki** değer. Öge sayısı çiftse ortadaki iki değer ortalamasıdır.

`ortanca` dalların **dışında** tanımlanmalıdır; yalnızca dalların içinde atanırsa derleyici **error: cannot find symbol ... variable ortanca** verir. `import java.util.Arrays;` gerekir.

Ortanca: Tek ve Çift Boyut



Girilen değerler sırasızdır: ortadaki hücre (9.0) ortanca **değildir**.

Adım 1 / 3

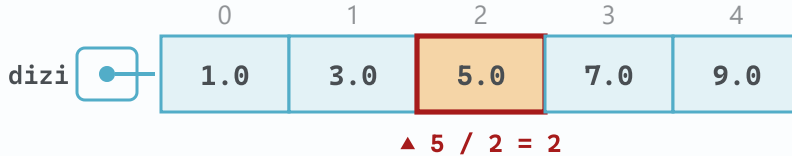
Neden sıralama?

Ortanca sıralamaya göre tanımlıdır. Sırasız dizinin ortasındaki öge rastgele bir değerdir.

Yan etki

`Arrays.sort` diziyi **yerinde** değiştirir; girilen sıra kaybolur. Sırayı korumak için önce `dizi.clone()` alınır.

Ortanca: Tek ve Çift Boyut



`Arrays.sort(dizi)` sonrası. Boyut 5 (tek): `dizi[5 / 2] = dizi[2]` → ortanca **5.0**.
Tam sayı bölmesi 2.5'i 2'ye keser.

Adım 2 / 3

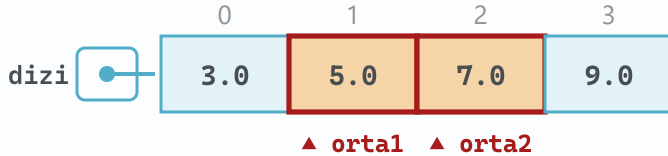
Neden sıralama?

Ortanca sıralamaya göre tanımlıdır. Sırasız dizinin ortasındaki öge rastgele bir değerdir.

Yan etki

`Arrays.sort` diziyi **yerinde** değiştirir; girilen sıra kaybolur. Sırayı korumak için önce `dizi.clone()` alınır.

Ortanca: Tek ve Çift Boyut



Boyut 4 (çift; girdiler 7 3 9 5): $\text{orta1} = 4/2 - 1 = 1$, $\text{orta2} = 4/2 = 2 \rightarrow (5.0 + 7.0) / 2 = 6.0$.

Adım 3 / 3

Neden sıralama?

Ortanca sıralamaya göre tanımlıdır. Sırasız dizinin ortasındaki öge rastgele bir değerdir.

Yan etki

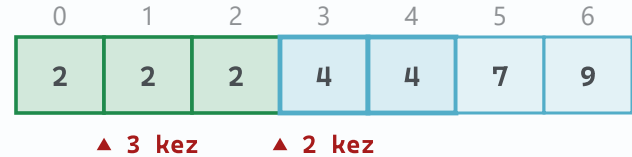
`Arrays.sort` diziyi **yerinde** değiştirir; girilen sıra kaybolur. Sırayı korumak için önce `dizi.clone()` alınır.

Mod Hesaplama

Mod (tepe değer)

Dizide **en sık tekrar eden** değer. 4 2 7 2 4 2 9 için mod 2'dir (3 kez).

- Önce `Arrays.sort(dizi)` : eşit değerler **yan yana** gelir.
- Sonra tek geçişte en uzun "seri" bulunur: her öge bir öncekiyle karşılaştırılır.



SONUCU YAZDIRMA VE ÇIKTI (GİRDİLER: 7, SONRA 4 2 7 2 4 2 9)

```
// Sonucu yazdır
System.out.println("Elemanlar sirali: " + Arrays.toString(dizi));
System.out.print("Mod: " + enCokTekrarEden);
System.out.println(" (Tekrar Sayisi: " + enCokTekrarSayisi + ")");
```

```
Elemanlar sirali: [2, 2, 2, 4, 4, 7, 9]
Mod: 2 (Tekrar Sayisi: 3)
```

Mod: Seri Sayma Adım Adım

```
▶ int enCokTekrarEden = dizi[0];
▶ int enCokTekrarSayisi = 1;
▶ int mevcutTekrarEden = dizi[0];
▶ int mevcutTekrarSayisi = 1;

5 for (int i = 1; i < boyut; i++) {
6     if (dizi[i] == dizi[i - 1]) {
7         mevcutTekrarSayisi++;
8     } else {
9         mevcutTekrarSayisi = 1;
10        mevcutTekrarEden = dizi[i];
11    }
12    if (mevcutTekrarSayisi > enCokTekrarSayisi) {
13        enCokTekrarSayisi = mevcutTekrarSayisi;
14        enCokTekrarEden = mevcutTekrarEden;
15    }
16 }
```

0	1	2	3	4	5	6
2	2	2	4	4	7	9

Dizi önceden **sıralandı**: eşit değerler artık yan yana. İlk öge hem mevcut hem en çok tekrar eden aday (1 kez).

Adım 1 / 8

DEĞİŞKENLER

mevcutTekrarEden = 2

mevcutTekrarSayisi = 1

enCokTekrarEden = 2

enCokTekrarSayisi = 1

Mod: Seri Sayma Adım Adım

```
1 int enCokTekrarEden = dizi[0];
2 int enCokTekrarSayisi = 1;
3 int mevcutTekrarEden = dizi[0];
4 int mevcutTekrarSayisi = 1;
5 for (int i = 1; i < boyut; i++) {
6     if (dizi[i] == dizi[i - 1]) {
7         ▶ mevcutTekrarSayisi++;
8     } else {
9         mevcutTekrarSayisi = 1;
10        mevcutTekrarEden = dizi[i];
11    }
12    if (mevcutTekrarSayisi > enCokTekrarSayisi) {
13        enCokTekrarSayisi = mevcutTekrarSayisi;
14        ▶ enCokTekrarEden = mevcutTekrarEden;
15    }
16 }
```

0	1	2	3	4	5	6
2	2	2	4	4	7	9

▲ i

dizi[1] = 2 == dizi[0] = 2 → seri devam ediyor, mevcutTekrarSayisi = 2 .
2 > önceki en iyi → **en çok tekrar eden 2 (2 kez).**

Adım 2 / 8

DEĞİŞKENLER

i = 1 mevcutTekrarEden = 2

mevcutTekrarSayisi = 2

enCokTekrarEden = 2

enCokTekrarSayisi = 2

Mod: Seri Sayma Adım Adım

```
1 int enCokTekrarEden = dizi[0];
2 int enCokTekrarSayisi = 1;
3 int mevcutTekrarEden = dizi[0];
4 int mevcutTekrarSayisi = 1;
5 for (int i = 1; i < boyut; i++) {
6     if (dizi[i] == dizi[i - 1]) {
7         ▶ mevcutTekrarSayisi++;
8     } else {
9         mevcutTekrarSayisi = 1;
10        mevcutTekrarEden = dizi[i];
11    }
12    if (mevcutTekrarSayisi > enCokTekrarSayisi) {
13        enCokTekrarSayisi = mevcutTekrarSayisi;
14        ▶ enCokTekrarEden = mevcutTekrarEden;
15    }
16 }
```

0	1	2	3	4	5	6
2	2	2	4	4	7	9

▲ i

dizi[2] = 2 == dizi[1] = 2 → seri devam ediyor, mevcutTekrarSayisi = 3 .
3 > önceki en iyi → **en çok tekrar eden 2 (3 kez)**.

Adım 3 / 8

DEĞİŞKENLER

i = 2 mevcutTekrarEden = 2

mevcutTekrarSayisi = 3

enCokTekrarEden = 2

enCokTekrarSayisi = 3

Mod: Seri Sayma Adım Adım

```
1 int enCokTekrarEden = dizi[0];
2 int enCokTekrarSayisi = 1;
3 int mevcutTekrarEden = dizi[0];
4 int mevcutTekrarSayisi = 1;
5 for (int i = 1; i < boyut; i++) {
6     if (dizi[i] == dizi[i - 1]) {
7         mevcutTekrarSayisi++;
8     } else {
9         mevcutTekrarSayisi = 1;
10        ► mevcutTekrarEden = dizi[i];
11    }
12    if (mevcutTekrarSayisi > enCokTekrarSayisi) {
13        enCokTekrarSayisi = mevcutTekrarSayisi;
14        enCokTekrarEden = mevcutTekrarEden;
15    }
16 }
```

0	1	2	3	4	5	6
2	2	2	4	4	7	9

▲ i

dizi[3] = 4 ≠ dizi[2] = 2 → yeni
seri başlıyor: mevcutTekrarEden = 4 ,
sayaç 1.

Adım 4 / 8

DEĞİŞKENLER

i = 3 mevcutTekrarEden = 4

mevcutTekrarSayisi = 1

enCokTekrarEden = 2

enCokTekrarSayisi = 3

Mod: Seri Sayma Adım Adım

```
1 int enCokTekrarEden = dizi[0];
2 int enCokTekrarSayisi = 1;
3 int mevcutTekrarEden = dizi[0];
4 int mevcutTekrarSayisi = 1;
5 for (int i = 1; i < boyut; i++) {
6     if (dizi[i] == dizi[i - 1]) {
7         ▶ mevcutTekrarSayisi++;
8     } else {
9         mevcutTekrarSayisi = 1;
10        mevcutTekrarEden = dizi[i];
11    }
12    if (mevcutTekrarSayisi > enCokTekrarSayisi) {
13        enCokTekrarSayisi = mevcutTekrarSayisi;
14        enCokTekrarEden = mevcutTekrarEden;
15    }
16 }
```

0	1	2	3	4	5	6
2	2	2	4	4	7	9

▲ i

dizi[4] = 4 == dizi[3] = 4 → seri devam ediyor, mevcutTekrarSayisi = 2 .

Adım 5 / 8

DEĞİŞKENLER

i = 4

mevcutTekrarEden = 4

mevcutTekrarSayisi = 2

enCokTekrarEden = 2

enCokTekrarSayisi = 3

Mod: Seri Sayma Adım Adım

```
1 int enCokTekrarEden = dizi[0];
2 int enCokTekrarSayisi = 1;
3 int mevcutTekrarEden = dizi[0];
4 int mevcutTekrarSayisi = 1;
5 for (int i = 1; i < boyut; i++) {
6     if (dizi[i] == dizi[i - 1]) {
7         mevcutTekrarSayisi++;
8     } else {
9         mevcutTekrarSayisi = 1;
10        mevcutTekrarEden = dizi[i];
11    }
12    if (mevcutTekrarSayisi > enCokTekrarSayisi) {
13        enCokTekrarSayisi = mevcutTekrarSayisi;
14        enCokTekrarEden = mevcutTekrarEden;
15    }
16 }
```

0	1	2	3	4	5	6
2	2	2	4	4	7	9

▲ i

dizi[5] = 7 ≠ dizi[4] = 4 → yeni
seri başlıyor: mevcutTekrarEden = 7,
sayaç 1.

Adım 6 / 8

DEĞİŞKENLER

i = 5 mevcutTekrarEden = 7

mevcutTekrarSayisi = 1

enCokTekrarEden = 2

enCokTekrarSayisi = 3

Mod: Seri Sayma Adım Adım

```
1 int enCokTekrarEden = dizi[0];
2 int enCokTekrarSayisi = 1;
3 int mevcutTekrarEden = dizi[0];
4 int mevcutTekrarSayisi = 1;
5 for (int i = 1; i < boyut; i++) {
6     if (dizi[i] == dizi[i - 1]) {
7         mevcutTekrarSayisi++;
8     } else {
9         mevcutTekrarSayisi = 1;
10        mevcutTekrarEden = dizi[i];
11    }
12    if (mevcutTekrarSayisi > enCokTekrarSayisi) {
13        enCokTekrarSayisi = mevcutTekrarSayisi;
14        enCokTekrarEden = mevcutTekrarEden;
15    }
16 }
```

0	1	2	3	4	5	6
2	2	2	4	4	7	9

▲ i

dizi[6] = 9 ≠ dizi[5] = 7 → yeni
seri başlıyor: mevcutTekrarEden = 9 ,
sayaç 1.

Adım 7 / 8

DEĞİŞKENLER

i = 6 mevcutTekrarEden = 9

mevcutTekrarSayisi = 1

enCokTekrarEden = 2

enCokTekrarSayisi = 3

Mod: Seri Sayma Adım Adım

```
1 int enCokTekrarEden = dizi[0];
2 int enCokTekrarSayisi = 1;
3 int mevcutTekrarEden = dizi[0];
4 int mevcutTekrarSayisi = 1;
5 ▶ for (int i = 1; i < boyut; i++) {
6     if (dizi[i] == dizi[i - 1]) {
7         mevcutTekrarSayisi++;
8     } else {
9         mevcutTekrarSayisi = 1;
10        mevcutTekrarEden = dizi[i];
11    }
12    if (mevcutTekrarSayisi > enCokTekrarSayisi) {
13        enCokTekrarSayisi = mevcutTekrarSayisi;
14        enCokTekrarEden = mevcutTekrarEden;
15    }
16 }
```

0	1	2	3	4	5	6
2	2	2	4	4	7	9

`i = 7` : döngü biter. **Mod = 2**, 3 kez tekrar ediyor. Eşit uzunlukta seriler olsaydı `>` kullanıldığı için **ilk** bulunan korunurdu.

Adım 8 / 8

DEĞİŞKENLER

`i = 7` `mevcutTekrarEden = 9`

`mevcutTekrarSayisi = 1`

`enCokTekrarEden = 2`

`enCokTekrarSayisi = 3`

Doğrusal (Lineer) Arama

Bir dizinin içinde belirli bir öğeyi aramanın en basit yolu: baştan sona **sırayla** her öğeye bakmak.

```
1 static boolean lineerArama(int[] dizi, int hedef)
2 {
3     for (int eleman : dizi) {
4         if (eleman == hedef) {
5             return true;
6         }
7     }
8     return false;
9 }
10 // main içinde:
11 int[] dizi = {10, 20, 30, 40, 50};
12 boolean sonuc = lineerArama(dizi, 30);
```

0	1	2	3	4
10	20	30	40	50

main, lineerArama(dizi, 30) 'u çağırır.
Metoda dizinin **referansı** ve 30 gider.

Adım 1 / 5

ÇAĞRI YIĞINI

lineerArama(dizi, 30)

main()

DEĞİŞKENLER

hedef = 30

Doğrusal (Lineer) Arama

Bir dizinin içinde belirli bir öğeyi aramanın en basit yolu: baştan sona **sırayla** her öğeye bakmak.

```
1 static boolean lineerArama(int[] dizi, int hedef)
  {
2     for (int eleman : dizi) {
3         if (eleman == hedef) {
4             return true;
5         }
6     }
7     return false;
8 }
9
10 // main içinde:
11 int[] dizi = {10, 20, 30, 40, 50};
12 boolean sonuc = lineerArama(dizi, 30);
```

0	1	2	3	4
10	20	30	40	50

▲ öğe

eleman = 10 : 10 == 30 yanlış → sıradaki öğeye geç.

Adım 2 / 5

ÇAĞRI YIĞINI

lineerArama(dizi, 30)

main()

DEĞİŞKENLER

hedef = 30

eleman = 10

Doğrusal (Lineer) Arama

Bir dizinin içinde belirli bir öğeyi aramanın en basit yolu: baştan sona **sırayla** her öğeye bakmak.

```
1 static boolean lineerArama(int[] dizi, int hedef)
2 {
3     for (int eleman : dizi) {
4         if (eleman == hedef) {
5             return true;
6         }
7     }
8     return false;
9 }
10 // main içinde:
11 int[] dizi = {10, 20, 30, 40, 50};
12 boolean sonuc = lineerArama(dizi, 30);
```

0	1	2	3	4
10	20	30	40	50

▲ öğe

eleman = 20 : 20 == 30 yanlış → sıradaki öğeye geç.

Adım 3 / 5

ÇAĞRI YIĞINI

lineerArama(dizi, 30)

main()

DEĞİŞKENLER

hedef = 30

eleman = 20

Doğrusal (Linear) Arama

Bir dizinin içinde belirli bir öğeyi aramanın en basit yolu: baştan sona **sırayla** her öğeye bakmak.

```
1 static boolean lineerArama(int[] dizi, int hedef)
  {
2     for (int eleman : dizi) {
3         if (eleman == hedef) {
4             return true;
5         }
6     }
7     return false;
8 }
9
10 // main içinde:
11 int[] dizi = {10, 20, 30, 40, 50};
12 boolean sonuc = lineerArama(dizi, 30);
```

0	1	2	3	4
10	20	30	40	50

▲ öğe

eleman = 30 : 30 == 30 → **doğru**. Döngü
yarıda kesilir, **true** döner.

Adım 4 / 5

ÇAĞRI YIĞINI

lineerArama(dizi, 30)

main()

DEĞİŞKENLER

hedef = 30

eleman = 30

Doğrusal (Lineer) Arama

Bir dizinin içinde belirli bir öğeyi aramanın en basit yolu: baştan sona **sırayla** her öğeye bakmak.

```
1 static boolean lineerArama(int[] dizi, int hedef)
2 {
3     for (int eleman : dizi) {
4         if (eleman == hedef) {
5             return true;
6         }
7     }
8     return false;
9 }
10 // main içinde:
11 int[] dizi = {10, 20, 30, 40, 50};
12 boolean sonuc = lineerArama(dizi, 30);
```

0	1	2	3	4
10	20	30	40	50

Metot çerçevesi yığından çıkar: **sonuc = true**. 3 karşılaştırma yetti; 40 ve 50'ye hiç bakılmadı. Hedef 35 olsaydı 5 karşılaştırmadan sonra **false** dönerdi.

Adım 5 / 5

ÇAĞRI YIĞINI

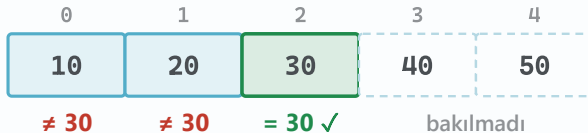
main()

DEĞİŞKENLER

sonuc = true

Doğrusal Aramanın Maliyeti

- En kötü durumda **tüm dizi** gezilir: n öğeli dizide n karşılaştırma.
- Zaman karmaşıklığı **$O(n)$** 'dir (n : dizinin boyutu); iş miktarı girdi boyutuyla **doğru orantılı** büyür.
- **İyi durum**: hedef erken bulunur (ör. ilk öğe), arama hızlı biter: 1 karşılaştırma.
- **Kötü durum**: hedef dizinin sonundadır ya da hiç yoktur, arama uzun sürer.
- Sırasız bir dizide daha iyisi yoktur: her öğeye bakmadan "yok" denemez.



aranan = 30 → 3 karşılaştırma; en kötü durum $n = 5$

```
int[] dizi = {10, 20, 30, 40, 50};  
// Bir dizide belirli bir elemanı arama  
int aranan = 30;  
boolean bulundu = false;  
for (int eleman : dizi) {  
    if (eleman == aranan) {  
        bulundu = true;  
        break;  
    }  
} // Bu işlem  $O(n)$  karmaşıklığına sahiptir.
```

break, hedef bulunduğu anda döngüden çıkar. Bu örnekte yalnızca 3 öğeye bakılır ve **bulundu = true** olur.

Eşleştirme: Ortak Öğeler

Çağrı: `eslestir(new int[]{3, 7, 1}, new int[]{5, 1, 3, 8})`

dizi2 →	5	1	3	8	
dizi1	3	≠	≠	=	≠
	7				
	1				

Dış döngü `eleman1 = 3` : iç döngü dizi2'nin **4 öğesinin hepsiyle** karşılaştırır. Eşleşme bulundu → yazdırılır.

Adım 1 / 4

```
1 static void eslestir(int[] dizi1, int[] dizi2) {  
2     for (int eleman1 : dizi1) {  
3         for (int eleman2 : dizi2) {  
4             if (eleman1 == eleman2) {  
5                 System.out.println("Eslesen Eleman: " + eleman1);  
6             }  
7         }  
8     }  
9 }
```

ÇIKTI

Eslesen Eleman: 3

Eşleştirme: Ortak Öğeler

Çağrı: `eslestir(new int[]{3, 7, 1}, new int[]{5, 1, 3, 8})`

dizi2 →	5	1	3	8	
dizi1	3	≠	≠	=	≠
	7	≠	≠	≠	≠
	1				

Dış döngü `eleman1 = 7` : iç döngü dizi2'nin **4 öğesinin hepsiyle** karşılaştırır. Eşleşme yok.

Adım 2 / 4

```
1 static void eslestir(int[] dizi1, int[] dizi2) {  
2     for (int eleman1 : dizi1) {  
3         for (int eleman2 : dizi2) {  
4             if (eleman1 == eleman2) {  
5                 System.out.println("Eslesen Eleman: " + eleman1);  
6             }  
7         }  
8     }  
9 }
```

ÇIKTI

Eslesen Eleman: 3

Eşleştirme: Ortak Öğeler

Çağrı: `eslestir(new int[]{3, 7, 1}, new int[]{5, 1, 3, 8})`

dizi2 →	5	1	3	8	
dizi1	3	≠	≠	=	≠
	7	≠	≠	≠	≠
	1	≠	=	≠	≠

Dış döngü `eleman1 = 1` : iç döngü dizi2'nin **4 öğesinin hepsiyle** karşılaştırır. Eşleşme bulundu → yazdırılır.

Adım 3 / 4

```
1 static void eslestir(int[] dizi1, int[] dizi2) {  
2     for (int eleman1 : dizi1) {  
3         for (int eleman2 : dizi2) {  
4             if (eleman1 == eleman2) {  
5                 System.out.println("Eslesen Eleman: " + eleman1);  
6             }  
7         }  
8     }  
9 }
```

ÇIKTI

```
Eslesen Eleman: 3  
Eslesen Eleman: 1
```

Eşleştirme: Ortak Öğeler

Çağrı: `eslestir(new int[]{3, 7, 1}, new int[]{5, 1, 3, 8})`

dizi2 →	5	1	3	8	
dizi1	3	≠	≠	=	≠
	7	≠	≠	≠	≠
	1	≠	=	≠	≠

Toplam $3 \times 4 = 12$ **karşılaştırma**. Diziler n öğeliyse $n \times n = n^2$ karşılaştırma yapılır.

Adım 4 / 4

```
1 static void eslestir(int[] dizi1, int[] dizi2) {  
2     for (int eleman1 : dizi1) {  
3         for (int eleman2 : dizi2) {  
4             if (eleman1 == eleman2) {  
5                 System.out.println("Eslesen Eleman: " + eleman1);  
6             }  
7         }  
8     }  
9 }
```

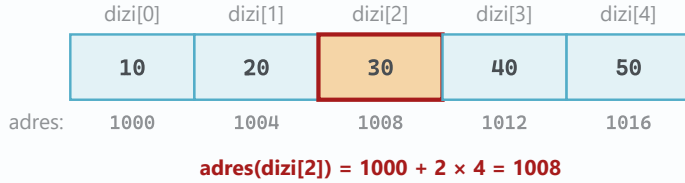
ÇIKTI

```
Eslesen Eleman: 3  
Eslesen Eleman: 1
```

DİZİ ALGORİTMALARI

Karmaşıklık: $O(1)$, $O(n)$, $O(n^2)$

NEDEN ERİŞİM $O(1)$?



Hücreler ardışık ve eşit boyutlu (`int` = 4 bayt) olduğu için i. öğenin adresi tek bir çarpma ve toplama ile **hesaplanır**: `adres = başlangıç + i × 4`. `dizi[2]` ile `dizi[1000000]` aynı sürede okunur.

İşlem	Karmaşıklık	n = 1000 için
Erişim <code>dizi[i]</code>	$O(1)$ sabit	1
Doğrusal arama	$O(n)$ lineer	≤ 1000
Tersine çevirme	$O(n)$	500 takas
Eşleştirme (iç içe)	$O(n^2)$ karesel	1 000 000
Seçimli sıralama	$O(n^2)$	$\approx 500\,000$

$O(n^2)$ algoritmalar büyük veri kümeleri için **verimsizdir**: n 10 kat büyürse süre 100 kat büyür. Erişim işlemi ise girdi boyundan bağımsızdır.

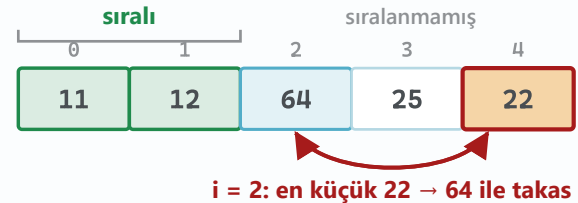
DİZİ ALGORİTMALARI

Seçimli Sıralama

```
static void secimliSiralama(int[] dizi) {  
    for (int i = 0; i < dizi.length - 1; i++) {  
        int minIndeks = i;  
        for (int j = i + 1; j < dizi.length; j++) {  
            if (dizi[j] < dizi[minIndeks]) {  
                minIndeks = j;  
            }  
        }  
        int gecici = dizi[minIndeks];  
        dizi[minIndeks] = dizi[i];  
        dizi[i] = gecici;  
    }  
}
```

- **Seçimli sıralama (Selection Sort):** her adımda en küçük öge bulunup sıralanmamış kısmın başına yerleştirilir.
- Dış döngü $n - 1$ tur, iç döngü kalan öğeleri tarar → $O(n^2)$ karşılaştırma, en fazla $n - 1$ takas.
- Takas **dizi içinde**, geçici değişkenle yapılır.

yerDegistir(dizi[minIndeks], dizi[i]) biçiminde bir metod **çalışmaz**: metoda öğelerin yalnızca **kopyaları** gider.



DİZİ ALGORİTMALARI · KODU ADIM ADIM ÇALIŞTIR

{64, 25, 12, 22, 11} Sıralanıyor



Başlangıç: hiçbir öge yerinde değil. Her turda kalan kısmın **en küçüğü** bulunup kalan kısmın **başına** konur.

Adım 1 / 9

```
int[] dizi = {64, 25, 12, 22, 11};
secimliSirala(dizi);
// secimliSirala her turun sonunda yazdırır:
// System.out.println("i=" + i + ": "
//                     + Arrays.toString(dizi));
```

DEĞİŞKENLER

i = -

minIndeks = -

ÇIKTI

DİZİ ALGORİTMALARI · KODU ADIM ADIM ÇALIŞTIR

{64, 25, 12, 22, 11} Sıralanıyor



Tur **i = 0** : iç döngü **j = 1...4** arasında en küçük değeri arar → **11** (indis 4).

Adım 2 / 9

```
int[] dizi = {64, 25, 12, 22, 11};
secimliSiralama(dizi);
// secimliSiralama her turun sonunda yazdırır:
// System.out.println("i=" + i + ": "
//                     + Arrays.toString(dizi));
```

DEĞİŞKENLER

i = 0

minIndeks = 4

ÇIKTI

DİZİ ALGORİTMALARI · KODU ADIM ADIM ÇALIŞTIR

{64, 25, 12, 22, 11} Sıralanıyor



`dizi[0]` ile `dizi[4]` yer değiştirdi. İlk 1 hücre artık **kesin yerinde**.

Adım 3 / 9

```
int[] dizi = {64, 25, 12, 22, 11};
secimliSiralama(dizi);
// secimliSiralama her turun sonunda yazdırır:
// System.out.println("i=" + i + ": "
//                     + Arrays.toString(dizi));
```

DEĞİŞKENLER

`i = 0`

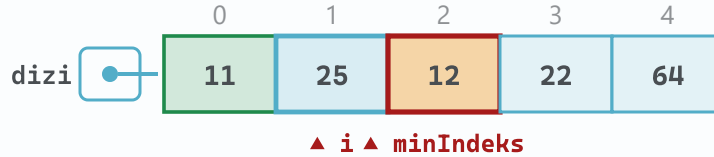
`minIndeks = 4`

ÇIKTI

`i=0: [11, 25, 12, 22, 64]`

DİZİ ALGORİTMALARI · KODU ADIM ADIM ÇALIŞTIR

{64, 25, 12, 22, 11} Sıralanıyor



Tur `i = 1` : iç döngü `j = 2...4` arasında en küçük değeri arar → **12** (indis 2).

Adım 4 / 9

```
int[] dizi = {64, 25, 12, 22, 11};
secimliSiralama(dizi);
// secimliSiralama her turun sonunda yazdırır:
// System.out.println("i=" + i + ": "
//                     + Arrays.toString(dizi));
```

DEĞİŞKENLER

`i = 1`

`minIndeks = 2`

ÇIKTI

`i=0: [11, 25, 12, 22, 64]`

DİZİ ALGORİTMALARI · KODU ADIM ADIM ÇALIŞTIR

{64, 25, 12, 22, 11} Sıralanıyor



`dizi[1]` ile `dizi[2]` yer değiştirdi. İlk 2 hücre artık **kesin yerinde**.

Adım 5 / 9

```
int[] dizi = {64, 25, 12, 22, 11};
secimliSirala(dizi);
// secimliSirala her turun sonunda yazdırır:
// System.out.println("i=" + i + ": "
//                     + Arrays.toString(dizi));
```

DEĞİŞKENLER

`i = 1`

`minIndeks = 2`

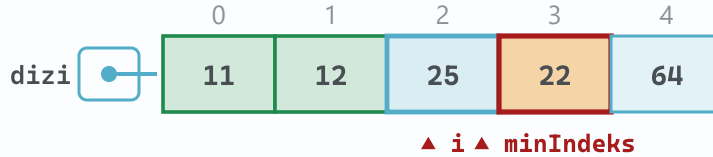
ÇIKTI

`i=0: [11, 25, 12, 22, 64]`

`i=1: [11, 12, 25, 22, 64]`

DİZİ ALGORİTMALARI · KODU ADIM ADIM ÇALIŞTIR

{64, 25, 12, 22, 11} Sıralanıyor



Tur **i = 2** : iç döngü **j = 3...4** arasında en küçük değeri arar → **22** (indis 3).

Adım 6 / 9

```
int[] dizi = {64, 25, 12, 22, 11};
secimliSiralama(dizi);
// secimliSiralama her turun sonunda yazdırır:
// System.out.println("i=" + i + ": "
//                     + Arrays.toString(dizi));
```

DEĞİŞKENLER

i = 2

minIndeks = 3

ÇIKTI

i=0: [11, 25, 12, 22, 64]

i=1: [11, 12, 25, 22, 64]

DİZİ ALGORİTMALARI · KODU ADIM ADIM ÇALIŞTIR

{64, 25, 12, 22, 11} Sıralanıyor



`dizi[2]` ile `dizi[3]` yer değiştirdi. İlk 3 hücre artık **kesin yerinde**.

Adım 7 / 9

```
int[] dizi = {64, 25, 12, 22, 11};
secimliSirala(dizi);
// secimliSirala her turun sonunda yazdırır:
// System.out.println("i=" + i + ": "
//                     + Arrays.toString(dizi));
```

DEĞİŞKENLER

`i = 2`

`minIndeks = 3`

ÇIKTI

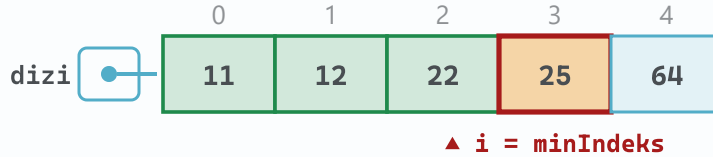
`i=0: [11, 25, 12, 22, 64]`

`i=1: [11, 12, 25, 22, 64]`

`i=2: [11, 12, 22, 25, 64]`

DİZİ ALGORİTMALARI · KODU ADIM ADIM ÇALIŞTIR

{64, 25, 12, 22, 11} Sıralanıyor



Tur $i = 3$: iç döngü $j = 4 \dots 4$ arasında en küçük değeri arar → 25 (indis 3).

Adım 8 / 9

```
int[] dizi = {64, 25, 12, 22, 11};
secimliSiralama(dizi);
// secimliSiralama her turun sonunda yazdırır:
// System.out.println("i=" + i + ": "
//                     + Arrays.toString(dizi));
```

DEĞİŞKENLER

$i = 3$

$\text{minIndeks} = 3$

ÇIKTI

```
i=0: [11, 25, 12, 22, 64]
i=1: [11, 12, 25, 22, 64]
i=2: [11, 12, 22, 25, 64]
```

DİZİ ALGORİTMALARI · KODU ADIM ADIM ÇALIŞTIR

{64, 25, 12, 22, 11} Sıralanıyor



En küçük zaten yerinde; kendisiyle takas edilir (değişiklik yok). 4 tur sonunda son öge da kendiliğinden yerindedir: dizi sıralı.

Adım 9 / 9

```
int[] dizi = {64, 25, 12, 22, 11};
secimliSiralama(dizi);
// secimliSiralama her turun sonunda yazdırır:
// System.out.println("i=" + i + ": "
//                     + Arrays.toString(dizi));
```

DEĞİŞKENLER

i = 3

minIndeks = 3

ÇIKTI

i=0: [11, 25, 12, 22, 64]

i=1: [11, 12, 25, 22, 64]

i=2: [11, 12, 22, 25, 64]

i=3: [11, 12, 22, 25, 64]

Neden yerDegistir(a, b) Çalışmaz?

```
// Çalışmaz: x ve y yalnızca kopyalardır
static void yerDegistir(int x, int y) {
    int gecici = x;
    x = y;
    y = gecici;
}

// Çalışır: dizinin kendisi (referansı) gelir
static void yerDegistir(int[] dizi, int i, int j) {
    int gecici = dizi[i];
    dizi[i] = dizi[j];
    dizi[j] = gecici;
}

public static void main(String[] args) {
    int[] dizi = {64, 25, 12};
    yerDegistir(dizi[0], dizi[2]);
    System.out.println(Arrays.toString(dizi));
    yerDegistir(dizi, 0, 2);
    System.out.println(Arrays.toString(dizi));
}
```

ÇIKTI

[64, 25, 12]

[12, 25, 64]

- Java'da parametreler **değerle geçirilir (pass by value)**. `yerDegistir(dizi[0], dizi[2])` çağrısında `x` ve `y`, 64 ve 12'nin **kopyalarıdır**; takas yalnızca kopyaları değiştirir, metot bitince kaybolur.
- İkinci metoda ise dizinin **referansı** (adresi) kopyalanır: metot aynı diziye ulaşır ve hücreleri gerçekten değiştirir.

Kopya mı, Referans mı? Çağrı Çerçeveleriyle

```
1 static void yerDegistir(int x, int y) {  
2     int gecici = x;  
3     x = y;  
4     y = gecici;  
5 }  
6 static void yerDegistir(int[] d, int i, int j) {  
7     int gecici = d[i];  
8     d[i] = d[j];  
9     d[j] = gecici;  
10 }  
▶ int[] dizi = {8, 3, 5}; // main  
12 yerDegistir(dizi[0], dizi[2]);  
13 System.out.println(Arrays.toString(dizi));  
14 yerDegistir(dizi, 0, 2);  
15 System.out.println(Arrays.toString(dizi));
```

Yığın (stack)

Heap



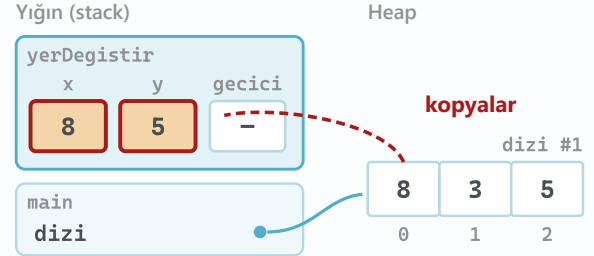
main'in çerçevesinde **dizi** referansı var;
dizinin kendisi heap'te.

Adım 1 / 13

DEĞİŞKENLER**dizi = →#1****ÇIKTI**

Kopya mı, Referans mı? Çağrı Çerçeveleriyle

```
► static void yerDegistir(int x, int y) {  
2     int gecici = x;  
3     x = y;  
4     y = gecici;  
5 }  
6 static void yerDegistir(int[] d, int i, int j) {  
7     int gecici = d[i];  
8     d[i] = d[j];  
9     d[j] = gecici;  
10 }  
11 int[] dizi = {8, 3, 5};           // main  
12 yerDegistir(dizi[0], dizi[2]);  
13 System.out.println(Arrays.toString(dizi));  
14 yerDegistir(dizi, 0, 2);  
15 System.out.println(Arrays.toString(dizi));
```



Çağrı: yığına yeni bir çerçeve eklenir. **x** ve **y** 'ye 8 ve 5'in **kopyaları** yazılır.

Adım 2 / 13

DEĞİŞKENLER

x = 8

y = 5

ÇIKTI

Kopya mı, Referans mı? Çağrı Çerçeveleriyle

```
1 static void yerDegistir(int x, int y) {  
2     int gecici = x;  
3     x = y;  
4     y = gecici;  
5 }  
6 static void yerDegistir(int[] d, int i, int j) {  
7     int gecici = d[i];  
8     d[i] = d[j];  
9     d[j] = gecici;  
10 }  
11 int[] dizi = {8, 3, 5};           // main  
12 yerDegistir(dizi[0], dizi[2]);  
13 System.out.println(Arrays.toString(dizi));  
14 yerDegistir(dizi, 0, 2);  
15 System.out.println(Arrays.toString(dizi));
```

Yığın (stack)

main
dizi

gecici = 8

Heap



Adım 3 / 13

DEĞİŞKENLER

x = 8

y = 5

gecici = 8

ÇIKTI



Kopya mı, Referans mı? Çağrı Çerçeveleriyle

```
1 static void yerDegistir(int x, int y) {  
2     int gecici = x;  
3     x = y;  
4     y = gecici;  
5 }  
6 static void yerDegistir(int[] d, int i, int j) {  
7     int gecici = d[i];  
8     d[i] = d[j];  
9     d[j] = gecici;  
10 }  
11 int[] dizi = {8, 3, 5};           // main  
12 yerDegistir(dizi[0], dizi[2]);  
13 System.out.println(Arrays.toString(dizi));  
14 yerDegistir(dizi, 0, 2);  
15 System.out.println(Arrays.toString(dizi));
```

Yığın (stack)

Heap



Adım 4 / 13

DEĞİŞKENLER

x = 5

y = 5

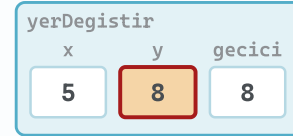
gecici = 8

ÇIKTI

Kopya mı, Referans mı? Çağrı Çerçeveleriyle

```
1 static void yerDegistir(int x, int y) {  
2     int gecici = x;  
3     x = y;  
4     y = gecici;  
5 }  
6 static void yerDegistir(int[] d, int i, int j) {  
7     int gecici = d[i];  
8     d[i] = d[j];  
9     d[j] = gecici;  
10 }  
11 int[] dizi = {8, 3, 5};           // main  
12 yerDegistir(dizi[0], dizi[2]);  
13 System.out.println(Arrays.toString(dizi));  
14 yerDegistir(dizi, 0, 2);  
15 System.out.println(Arrays.toString(dizi));
```

Yığın (stack)



Heap



y = 8 : x ile y takas oldu, ama dizideki hücreler hiç değişmedi.

Adım 5 / 13

DEĞİŞKENLER

x = 5 y = 8 gecici = 8

ÇIKTI



Kopya mı, Referans mı? Çağrı Çerçeveleriyle

```
1 static void yerDegistir(int x, int y) {  
2     int gecici = x;  
3     x = y;  
4     y = gecici;  
5 }  
6 static void yerDegistir(int[] d, int i, int j) {  
7     int gecici = d[i];  
8     d[i] = d[j];  
9     d[j] = gecici;  
10 }  
11 int[] dizi = {8, 3, 5};           // main  
12 yerDegistir(dizi[0], dizi[2]);  
13 System.out.println(Arrays.toString(dizi));  
14 yerDegistir(dizi, 0, 2);  
15 System.out.println(Arrays.toString(dizi));
```

Yığın (stack)

Heap



Metot bitti, çerçevesi silindi; kopyalar da kayboldu. dizi hâlâ {8, 3, 5}.

Adım 6 / 13

DEĞİŞKENLER

dizi = →#1

ÇIKTI

Kopya mı, Referans mı? Çağrı Çerçeveleriyle

```
1 static void yerDegistir(int x, int y) {  
2     int gecici = x;  
3     x = y;  
4     y = gecici;  
5 }  
6 static void yerDegistir(int[] d, int i, int j) {  
7     int gecici = d[i];  
8     d[i] = d[j];  
9     d[j] = gecici;  
10 }  
11 int[] dizi = {8, 3, 5};           // main  
12 yerDegistir(dizi[0], dizi[2]);  
13 ▶ System.out.println(Arrays.toString(dizi));  
14 yerDegistir(dizi, 0, 2);  
15 System.out.println(Arrays.toString(dizi));
```

Yığın (stack)

Heap



Adım 7 / 13

DEĞİŞKENLER

dizi = →#1

ÇIKTI

[8, 3, 5]

Kopya mı, Referans mı? Çağrı Çerçeveleriyle

```
1 static void yerDegistir(int x, int y) {  
2     int gecici = x;  
3     x = y;  
4     y = gecici;  
5 }  
▶ static void yerDegistir(int[] d, int i, int j) {  
7     int gecici = d[i];  
8     d[i] = d[j];  
9     d[j] = gecici;  
10 }  
11 int[] dizi = {8, 3, 5};           // main  
12 yerDegistir(dizi[0], dizi[2]);  
13 System.out.println(Arrays.toString(dizi));  
14 yerDegistir(dizi, 0, 2);  
15 System.out.println(Arrays.toString(dizi));
```



Adım 8 / 13

DEĞİŞKENLER

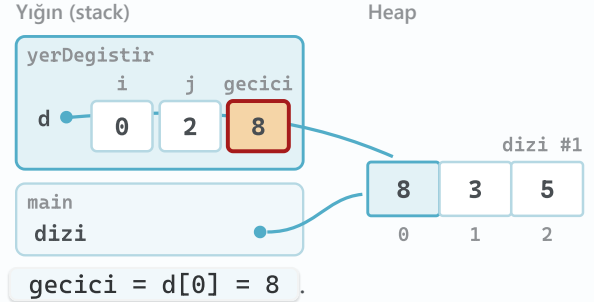
`d = →#1``i = 0``j = 2`

ÇIKTI

`[8, 3, 5]`

Kopya mı, Referans mı? Çağrı Çerçeveleriyle

```
1 static void yerDegistir(int x, int y) {  
2     int gecici = x;  
3     x = y;  
4     y = gecici;  
5 }  
6 static void yerDegistir(int[] d, int i, int j) {  
7     int gecici = d[i];  
8     d[i] = d[j];  
9     d[j] = gecici;  
10 }  
11 int[] dizi = {8, 3, 5};           // main  
12 yerDegistir(dizi[0], dizi[2]);  
13 System.out.println(Arrays.toString(dizi));  
14 yerDegistir(dizi, 0, 2);  
15 System.out.println(Arrays.toString(dizi));
```



Adım 9 / 13

DEĞİŞKENLER

d = →#1

i = 0

j = 2

gecici = 8

ÇIKTI

[8, 3, 5]

Kopya mı, Referans mı? Çağrı Çerçeveleriyle

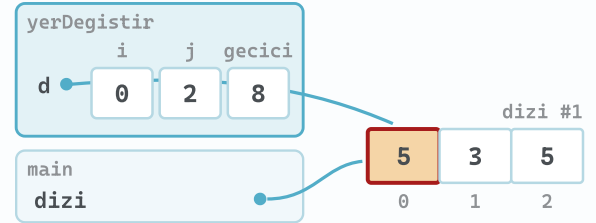
```

1 static void yerDegistir(int x, int y) {
2     int gecici = x;
3     x = y;
4     y = gecici;
5 }
6 static void yerDegistir(int[] d, int i, int j) {
7     int gecici = d[i];
8     d[i] = d[j];
9     d[j] = gecici;
10 }
11 int[] dizi = {8, 3, 5};           // main
12 yerDegistir(dizi[0], dizi[2]);
13 System.out.println(Arrays.toString(dizi));
14 yerDegistir(dizi, 0, 2);
15 System.out.println(Arrays.toString(dizi));

```

Yığın (stack)

Heap



`d[0] = d[2]` : heap'teki gerçek hücre 5 oldu.

Adım 10 / 13

DEĞİŞKENLER

d = →#1

i = 0

j = 2

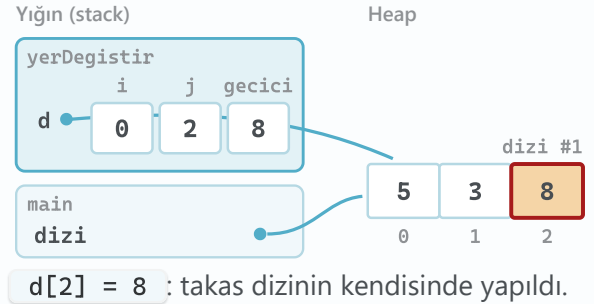
gecici = 8

ÇIKTI

[8, 3, 5]

Kopya mı, Referans mı? Çağrı Çerçeveleriyle

```
1 static void yerDegistir(int x, int y) {  
2     int gecici = x;  
3     x = y;  
4     y = gecici;  
5 }  
6 static void yerDegistir(int[] d, int i, int j) {  
7     int gecici = d[i];  
8     d[i] = d[j];  
9     d[j] = gecici;  
10 }  
11 int[] dizi = {8, 3, 5};           // main  
12 yerDegistir(dizi[0], dizi[2]);  
13 System.out.println(Arrays.toString(dizi));  
14 yerDegistir(dizi, 0, 2);  
15 System.out.println(Arrays.toString(dizi));
```



Adım 11 / 13

DEĞİŞKENLER

d = →#1

i = 0

j = 2

gecici = 8

ÇIKTI

[8, 3, 5]

Kopya mı, Referans mı? Çağrı Çerçeveleriyle

```
1 static void yerDegistir(int x, int y) {  
2     int gecici = x;  
3     x = y;  
4     y = gecici;  
5 }  
6 static void yerDegistir(int[] d, int i, int j) {  
7     int gecici = d[i];  
8     d[i] = d[j];  
9     d[j] = gecici;  
10 }  
11 int[] dizi = {8, 3, 5};           // main  
12 yerDegistir(dizi[0], dizi[2]);  
13 System.out.println(Arrays.toString(dizi));  
14 ► yerDegistir(dizi, 0, 2);  
15 System.out.println(Arrays.toString(dizi));
```

Yığın (stack)

Heap



Çerçeve silindi, ama değişiklik heap'te kaldı:
main aynı diziyi görüyor.

Adım 12 / 13

DEĞİŞKENLER

dizi = →#1

ÇIKTI

[8, 3, 5]

Kopya mı, Referans mı? Çağrı Çerçeveleriyle

```
1 static void yerDegistir(int x, int y) {  
2     int gecici = x;  
3     x = y;  
4     y = gecici;  
5 }  
6 static void yerDegistir(int[] d, int i, int j) {  
7     int gecici = d[i];  
8     d[i] = d[j];  
9     d[j] = gecici;  
10 }  
11 int[] dizi = {8, 3, 5};           // main  
12 yerDegistir(dizi[0], dizi[2]);  
13 System.out.println(Arrays.toString(dizi));  
14 yerDegistir(dizi, 0, 2);  
▶ System.out.println(Arrays.toString(dizi));
```

Yığın (stack)

Heap



Adım 13 / 13

DEĞİŞKENLER

dizi = →#1

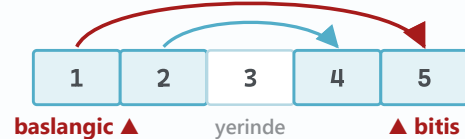
ÇIKTI

```
[8, 3, 5]  
[5, 3, 8]
```

Diziyi Tersine Çevirme

```
static void tersineCevir(int[] dizi) {  
    int baslangic = 0;  
    int bitis = dizi.length - 1;  
    while (baslangic < bitis) {  
        // Başlangıç ve bitiş elemanlarını değiştir  
        int gecici = dizi[baslangic];  
        dizi[baslangic] = dizi[bitis];  
        dizi[bitis] = gecici;  
        // Başlangıç indeksini artır, bitiş indeksini azalt  
        baslangic++;  
        bitis--;  
    }  
}
```

- **İki işaretçi (two pointers)** tekniği: biri baştan, biri sondan gelir; her turda uçtaki iki öge takas edilir.
- Yaklaşık $n/2$ takas → **$O(n)$** ; ek dizi gerekmez (yerinde, in-place).
- Metot `void` 'dir, ama dizi referansla geldiği için **çağırının dizisi** değişir.

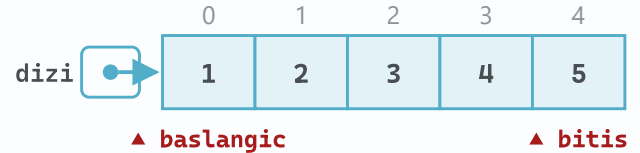


ÇIKTI

```
[1, 2, 3, 4, 5]  
→ [5, 4, 3, 2, 1]  
[1, 2, 3, 4]  
→ [4, 3, 2, 1]
```

tersineCevir: Adım Adım

```
1 static void tersineCevir(int[] dizi) {  
2     int baslangic = 0;  
3     int bitis = dizi.length - 1;  
4     while (baslangic < bitis) {  
5         int gecici = dizi[baslangic];  
6         dizi[baslangic] = dizi[bitis];  
7         dizi[bitis] = gecici;  
8         baslangic++;  
9         bitis--;  
10    }  
11 }
```



İki işaretçi: `baslangic = 0` soldan, `bitis = length - 1 = 4` sağdan başlar.

Adım 1 / 4

ÇAĞRI YIĞINI

`tersineCevir(dizi)`

`main()`

DEĞİŞKENLER

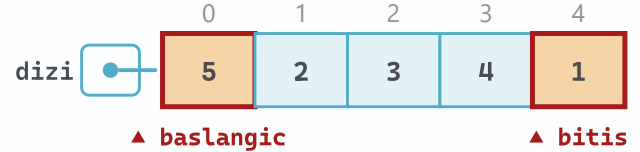
`baslangic = 0`

`bitis = 4`

`dizi = [1, 2, 3, 4, 5]`

tersineCevir: Adım Adım

```
1 static void tersineCevir(int[] dizi) {  
2     int baslangic = 0;  
3     int bitis = dizi.length - 1;  
4     while (baslangic < bitis) {  
5         ▶ int gecici = dizi[baslangic];  
6         ▶ dizi[baslangic] = dizi[bitis];  
7         ▶ dizi[bitis] = gecici;  
8         baslangic++;  
9         bitis--;  
10    }  
11 }
```



0 < 4 → **gecici** yardımıyla **dizi[0]** ile **dizi[4]** yer değiştirir (**gecici = 1**).

Adım 2 / 4

ÇAĞRI YIĞINI

tersineCevir(dizi)

main()

DEĞİŞKENLER

baslangic = 0

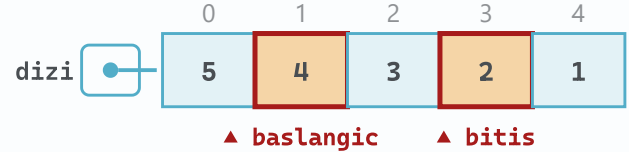
bitis = 4

gecici = 1

dizi = [5, 2, 3, 4, 1]

tersineCevir: Adım Adım

```
1 static void tersineCevir(int[] dizi) {  
2     int baslangic = 0;  
3     int bitis = dizi.length - 1;  
4     while (baslangic < bitis) {  
5         int gecici = dizi[baslangic];  
6         dizi[baslangic] = dizi[bitis];  
7         dizi[bitis] = gecici;  
8         baslangic++;  
9         bitis--;  
10    }  
11 }
```



1 < 3 → gecici yardımıyla dizi[1] ile dizi[3] yer değiştirir (gecici = 2).

Adım 3 / 4

ÇAĞRI YIĞINI

tersineCevir(dizi)

main()

DEĞİŞKENLER

baslangic = 1

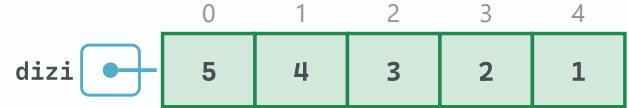
bitis = 3

gecici = 2

dizi = [5, 4, 3, 2, 1]

tersineCevir: Adım Adım

```
1 static void tersineCevir(int[] dizi) {  
2     int baslangic = 0;  
3     int bitis = dizi.length - 1;  
4     while (baslangic < bitis) {  
5         int gecici = dizi[baslangic];  
6         dizi[baslangic] = dizi[bitis];  
7         dizi[bitis] = gecici;  
8         baslangic++;  
9         bitis--;  
10    }  
11 }
```



▲ **baslangic = bitis**

baslangic = bitis = 2 : $2 < 2$ yanlış → döngü biter; ortadaki öge yerinde kalır. Metot hiçbir şey döndürmez ama **çağırının dizisi** değişmiştir: [5, 4, 3, 2, 1].

Adım 4 / 4

ÇAĞRI YİĞİNİ

tersineCevir(dizi)

main()

DEĞİŞKENLER

baslangic = 2

bitis = 2

dizi = [5, 4, 3, 2, 1]

ÇOK BOYUTLU DİZİLER

İki Boyutlu Diziler (Matrisler)

- Çok boyutlu diziler, birden fazla boyutta veri saklamak için kullanılır; Java iki (matris) ve daha fazla boyutlu dizileri destekler.
- İki boyutlu dizi, **satır ve sütunlarla** tanımlanan tablo benzeri bir yapıdır: satranç tahtası, resim, not çizelgesi.
- Erişim: `matris[satır][sutun]` ; ilk indis **satır**, ikincisi **sütun**.

```
int[][] matris = new int[3][3]; // 3x3 boyutunda bir matris  
matris[1][2] = 6; // 2. satır, 3. sütun
```

	satır ↓ sütun →		
	0	1	2
0	0	0	0
1	0	0	6
2	0	0	0

ÇOK BOYUTLU DİZİLER

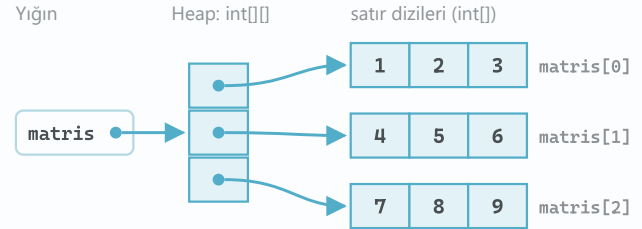
İlk Değer Verme ve Diziler Dizisi

- İki boyutlu dizilere de ilklendirici listesiyle başlangıç değerleri atanabilir; boyut belirtmeye gerek yoktur.
- Java'da 2B dizi aslında **dizilerin dizisidir**:
`matris[i]` i. satırın kendisi, yani bir `int[]` 'dir.
- `matris.length` = **satır** sayısı,
`matris[0].length` = ilk satırın **sütun** sayısı.

```
int[][] matris = { {1, 2, 3}, {4, 5, 6}, {7, 8, 9} };  
System.out.println(matris.length + " "  
    + matris[0].length + " " + matris[1][2]);
```

ÇIKTI

```
3 3 6
```

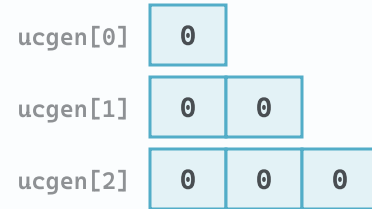


Düzensiz (Ragged) Diziler

```
int[][] ucgen = new int[3][];  
for (int i = 0; i < ucgen.length; i++) {  
    ucgen[i] = new int[i + 1];  
}  
System.out.println(ucgen[0].length + " " + ucgen[1].length  
    + " " + ucgen[2].length);  
int[][] tablo = {{1, 2}, {3, 4, 5}};  
for (int[] satir : tablo) {  
    for (int x : satir) {  
        System.out.print(x + " ");  
    }  
    System.out.println();  
}
```

ÇIKTI

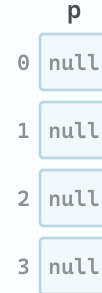
```
1 2 3  
1 2  
3 4 5
```



- Satırlar ayrı diziler olduğu için **uzunlukları farklı** olabilir.
- `new int[3][]` yalnızca satır referanslarını oluşturur (hepsi `null`); satırlar sonra ayrılır.
- Bu yüzden iç döngüde `matris[i].length` kullanmak, sabit sütun sayısından daha güvenlidir.
- `for-each` ile: dış döngü ögesi bir `int[]` satırdır.

Pascal Üçgeni: Satır Satır Düzensiz Dizi

```
► int[][] p = new int[4][];  
2 for (int i = 0; i < p.length; i++) {  
3     p[i] = new int[i + 1];  
4     p[i][0] = 1;  
5     p[i][i] = 1;  
6     for (int j = 1; j < i; j++) {  
7         p[i][j] = p[i - 1][j - 1] + p[i - 1][j];  
8     }  
9 }  
10 System.out.println(Arrays.toString(p[3]));
```



`new int[4][]` yalnızca 4 satır referansı ayırır;
hepsi `null`.

Adım 1 / 13

DEĞİŞKENLER

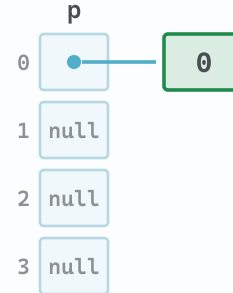
i = -

ÇIKTI



Pascal Üçgeni: Satır Satır Düzensiz Dizi

```
1 int[][] p = new int[4][];  
2 for (int i = 0; i < p.length; i++) {  
3     p[i] = new int[i + 1];  
4     p[i][0] = 1;  
5     p[i][i] = 1;  
6     for (int j = 1; j < i; j++) {  
7         p[i][j] = p[i - 1][j - 1] + p[i - 1][j];  
8     }  
9 }  
10 System.out.println(Arrays.toString(p[3]));
```



`i = 0`: 1 hücreli satır oluşur (0 ile dolu), `p[0]` onu gösterir.

Adım 2 / 13

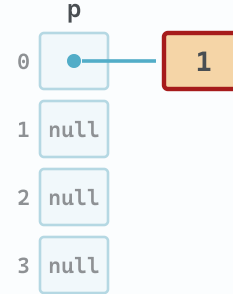
DEĞİŞKENLER

`i = 0`

ÇIKTI

Pascal Üçgeni: Satır Satır Düzensiz Dizi

```
1 int[][] p = new int[4][];  
2 for (int i = 0; i < p.length; i++) {  
3     p[i] = new int[i + 1];  
4     p[i][0] = 1;  
5     p[i][i] = 1;  
6     for (int j = 1; j < i; j++) {  
7         p[i][j] = p[i - 1][j - 1] + p[i - 1][j];  
8     }  
9 }  
10 System.out.println(Arrays.toString(p[3]));
```



Satırın ilk ve son hücresi aynı hücre: 1.

Adım 3 / 13

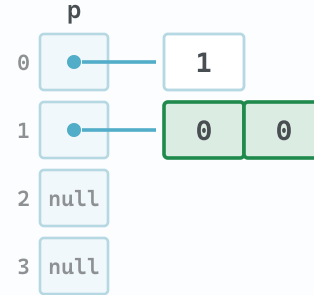
DEĞİŞKENLER

`i = 0`

ÇIKTI

Pascal Üçgeni: Satır Satır Düzensiz Dizi

```
1 int[][] p = new int[4][];  
2 for (int i = 0; i < p.length; i++) {  
3     p[i] = new int[i + 1];  
4     p[i][0] = 1;  
5     p[i][i] = 1;  
6     for (int j = 1; j < i; j++) {  
7         p[i][j] = p[i - 1][j - 1] + p[i - 1][j];  
8     }  
9 }  
10 System.out.println(Arrays.toString(p[3]));
```



i = 1: 2 hücreli satır.

Adım 4 / 13

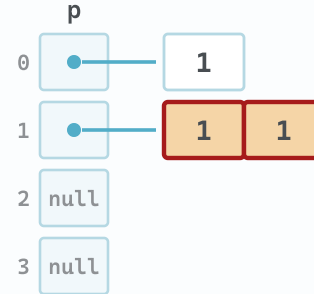
DEĞİŞKENLER

i = 1

ÇIKTI

Pascal Üçgeni: Satır Satır Düzensiz Dizi

```
1 int[][] p = new int[4][];  
2 for (int i = 0; i < p.length; i++) {  
3     p[i] = new int[i + 1];  
4     p[i][0] = 1;  
5     p[i][i] = 1;  
6     for (int j = 1; j < i; j++) {  
7         p[i][j] = p[i - 1][j - 1] + p[i - 1][j];  
8     }  
9 }  
10 System.out.println(Arrays.toString(p[3]));
```



Uçlar 1; iç hücre yok (`j < 1` hemen yanlış).

Adım 5 / 13

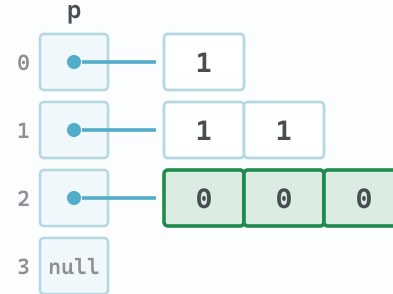
DEĞİŞKENLER

`i = 1`

ÇIKTI

Pascal Üçgeni: Satır Satır Düzensiz Dizi

```
1 int[][] p = new int[4][];  
2 for (int i = 0; i < p.length; i++) {  
3     p[i] = new int[i + 1];  
4     p[i][0] = 1;  
5     p[i][i] = 1;  
6     for (int j = 1; j < i; j++) {  
7         p[i][j] = p[i - 1][j - 1] + p[i - 1][j];  
8     }  
9 }  
10 System.out.println(Arrays.toString(p[3]));
```



i = 2: 3 hücreli satır.

Adım 6 / 13

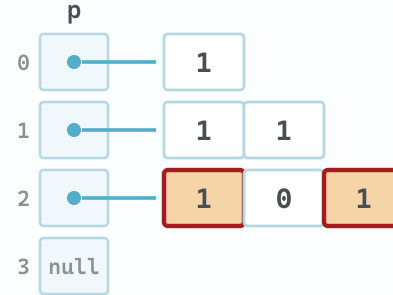
DEĞİŞKENLER

i = 2

ÇIKTI

Pascal Üçgeni: Satır Satır Düzensiz Dizi

```
1 int[][] p = new int[4][];  
2 for (int i = 0; i < p.length; i++) {  
3     p[i] = new int[i + 1];  
4     p[i][0] = 1;  
5     p[i][i] = 1;  
6     for (int j = 1; j < i; j++) {  
7         p[i][j] = p[i - 1][j - 1] + p[i - 1][j];  
8     }  
9 }  
10 System.out.println(Arrays.toString(p[3]));
```



Uçlar 1.

Adım 7 / 13

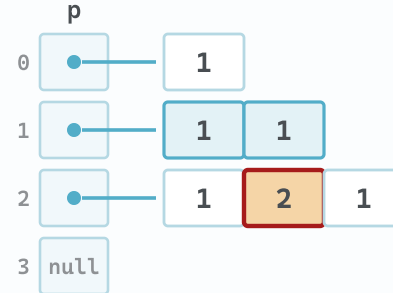
DEĞİŞKENLER

i = 2

ÇIKTI

Pascal Üçgeni: Satır Satır Düzensiz Dizi

```
1 int[][] p = new int[4][];  
2 for (int i = 0; i < p.length; i++) {  
3     p[i] = new int[i + 1];  
4     p[i][0] = 1;  
5     p[i][i] = 1;  
6     for (int j = 1; j < i; j++) {  
7         ► p[i][j] = p[i - 1][j - 1] + p[i - 1][j];  
8     }  
9 }  
10 System.out.println(Arrays.toString(p[3]));
```



$p[2][1] = p[1][0] + p[1][1] = 1 + 1 = 2$

Adım 8 / 13

DEĞİŞKENLER

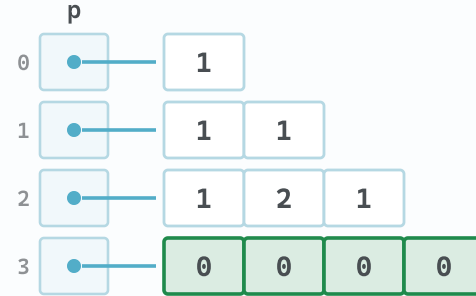
i = 2

j = 1

ÇIKTI

Pascal Üçgeni: Satır Satır Düzensiz Dizi

```
1 int[][] p = new int[4][];  
2 for (int i = 0; i < p.length; i++) {  
3     p[i] = new int[i + 1];  
4     p[i][0] = 1;  
5     p[i][i] = 1;  
6     for (int j = 1; j < i; j++) {  
7         p[i][j] = p[i - 1][j - 1] + p[i - 1][j];  
8     }  
9 }  
10 System.out.println(Arrays.toString(p[3]));
```



i = 3: 4 hücreli satır.

Adım 9 / 13

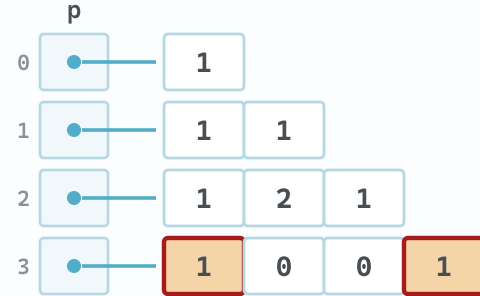
DEĞİŞKENLER

i = 3

ÇIKTI

Pascal Üçgeni: Satır Satır Düzensiz Dizi

```
1 int[][] p = new int[4][];  
2 for (int i = 0; i < p.length; i++) {  
3     p[i] = new int[i + 1];  
4     p[i][0] = 1;  
5     p[i][i] = 1;  
6     for (int j = 1; j < i; j++) {  
7         p[i][j] = p[i - 1][j - 1] + p[i - 1][j];  
8     }  
9 }  
10 System.out.println(Arrays.toString(p[3]));
```



Uçlar 1.

Adım 10 / 13

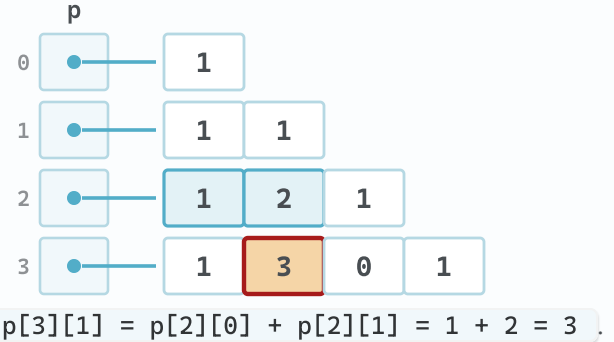
DEĞİŞKENLER

i = 3

ÇIKTI

Pascal Üçgeni: Satır Satır Düzensiz Dizi

```
1 int[][] p = new int[4][];  
2 for (int i = 0; i < p.length; i++) {  
3     p[i] = new int[i + 1];  
4     p[i][0] = 1;  
5     p[i][i] = 1;  
6     for (int j = 1; j < i; j++) {  
7         ► p[i][j] = p[i - 1][j - 1] + p[i - 1][j];  
8     }  
9 }  
10 System.out.println(Arrays.toString(p[3]));
```



Adım 11 / 13

DEĞİŞKENLER

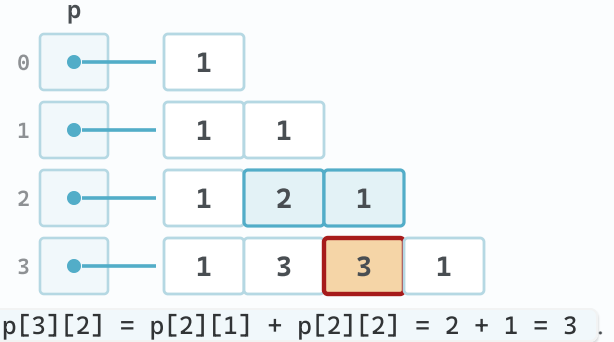
i = 3

j = 1

ÇIKTI

Pascal Üçgeni: Satır Satır Düzensiz Dizi

```
1 int[][] p = new int[4][];  
2 for (int i = 0; i < p.length; i++) {  
3     p[i] = new int[i + 1];  
4     p[i][0] = 1;  
5     p[i][i] = 1;  
6     for (int j = 1; j < i; j++) {  
7         ► p[i][j] = p[i - 1][j - 1] + p[i - 1][j];  
8     }  
9 }  
10 System.out.println(Arrays.toString(p[3]));
```



Adım 12 / 13

DEĞİŞKENLER

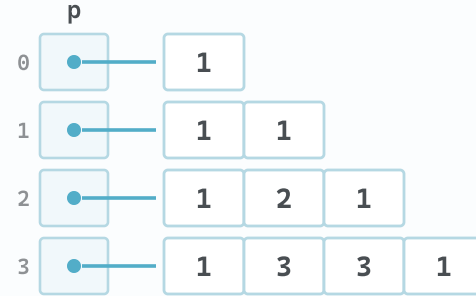
i = 3

j = 2

ÇIKTI

Pascal Üçgeni: Satır Satır Düzensiz Dizi

```
1 int[][] p = new int[4][];  
2 for (int i = 0; i < p.length; i++) {  
3     p[i] = new int[i + 1];  
4     p[i][0] = 1;  
5     p[i][i] = 1;  
6     for (int j = 1; j < i; j++) {  
7         p[i][j] = p[i - 1][j - 1] + p[i - 1][j];  
8     }  
9 }  
▶ System.out.println(Arrays.toString(p[3]));
```



Satırların uzunlukları 1, 2, 3, 4: düzensiz dizi. Son satır [1, 3, 3, 1] yazılır.

Adım 13 / 13

DEĞİŞKENLER

—

ÇIKTI

[1, 3, 3, 1]

ÇOK BOYUTLU DİZİLER

Örnek: ÇokBoyutluDiziOrnegi

```
public class ÇokBoyutluDiziOrnegi {  
    public static void main(String[] args) {  
        // 3x3 boyutunda iki boyutlu bir dizi (matris)  
        int[][] matris = new int[3][3];  
        // Matrisin elemanlarına ilk değerlerini atama  
        for (int satir = 0; satir < 3; satir++) {  
            for (int sutun = 0; sutun < 3; sutun++) {  
                matris[satir][sutun] = satir * 3 + sutun + 1;  
            }  
        }  
        // Matrisi ekrana yazdırma  
        System.out.println("Matrisin Icerigi:");  
        for (int satir = 0; satir < 3; satir++) {  
            for (int sutun = 0; sutun < 3; sutun++) {  
                System.out.print(matris[satir][sutun] + " ");  
            }  
            System.out.println(); // Yeni satıra geçme  
        }  
    }  
}
```

ÇIKTI

Matrisin Icerigi:

```
1 2 3  
4 5 6  
7 8 9
```

İç içe döngü

Dış döngü **satırları**, iç döngü bir satırın **sütunlarını** dolaşır. **satir** * 3 + **sutun** + 1 hücreleri 1'den 9'a numaralar.

Matris Doldurma ve Yazdırma: Adım Adım

```
► int[][] matris = new int[3][3];
2 for (int satir = 0; satir < 3; satir++) {
3     for (int sutun = 0; sutun < 3; sutun++) {
4         matris[satir][sutun] = satir * 3 + sutun +
5         1;
6     }
7 System.out.println("Matrisin Icerigi:");
8 for (int satir = 0; satir < 3; satir++) {
9     for (int sutun = 0; sutun < 3; sutun++) {
10        System.out.print(matris[satir][sutun] + "
11        ");
12    }
13    System.out.println();
14 }
```

	0	1	2
0	0	0	0
1	0	0	0
2	0	0	0

`new int[3][3]` : 3 satır × 3 sütun, tüm hücreler 0.

Adım 1 / 11

DEĞİŞKENLER

—

ÇIKTI

Matris Doldurma ve Yazdırma: Adım Adım

```
1 int[][] matris = new int[3][3];
2 for (int satir = 0; satir < 3; satir++) {
3     for (int sutun = 0; sutun < 3; sutun++) {
4         matris[satir][sutun] = satir * 3 + sutun +
5         1;
6     }
7 }
8 System.out.println("Matrisin Icerigi:");
9 for (int satir = 0; satir < 3; satir++) {
10     for (int sutun = 0; sutun < 3; sutun++) {
11         System.out.print(matris[satir][sutun] + "
12 ");
13     }
14     System.out.println();
15 }
```

	0	1	2
0	1	0	0
1	0	0	0
2	0	0	0

$\text{matris}[0][0] = 0 * 3 + 0 + 1 = 1$

Adım 2 / 11

DEĞİŞKENLER

satir = 0

sutun = 0

ÇIKTI

Matris Doldurma ve Yazdırma: Adım Adım

```
1 int[][] matris = new int[3][3];
2 for (int satir = 0; satir < 3; satir++) {
3     for (int sutun = 0; sutun < 3; sutun++) {
4         matris[satir][sutun] = satir * 3 + sutun +
5         1;
6     }
7 }
8 System.out.println("Matrisin Icerigi:");
9 for (int satir = 0; satir < 3; satir++) {
10     for (int sutun = 0; sutun < 3; sutun++) {
11         System.out.print(matris[satir][sutun] + "
12 ");
13     }
14     System.out.println();
15 }
```

	0	1	2
0	1	2	0
1	0	0	0
2	0	0	0

$\text{matris}[0][1] = 0 * 3 + 1 + 1 = 2$

Adım 3 / 11

DEĞİŞKENLER

satir = 0

sutun = 1

ÇIKTI

Matris Doldurma ve Yazdırma: Adım Adım

```
1 int[][] matris = new int[3][3];
2 for (int satir = 0; satir < 3; satir++) {
3     for (int sutun = 0; sutun < 3; sutun++) {
4         matris[satir][sutun] = satir * 3 + sutun +
5         1;
6     }
7 }
8 System.out.println("Matrisin Icerigi:");
9 for (int satir = 0; satir < 3; satir++) {
10     for (int sutun = 0; sutun < 3; sutun++) {
11         System.out.print(matris[satir][sutun] + "
12 ");
13     }
14 }
15 System.out.println();
```

	0	1	2
0	1	2	3
1	0	0	0
2	0	0	0

`matris[0][2] = 0 * 3 + 2 + 1 = 3`

Adım 4 / 11

DEĞİŞKENLER

satir = 0

sutun = 2

ÇIKTI

Matris Doldurma ve Yazdırma: Adım Adım

```
1 int[][] matris = new int[3][3];
2 for (int satir = 0; satir < 3; satir++) {
3     for (int sutun = 0; sutun < 3; sutun++) {
4         matris[satir][sutun] = satir * 3 + sutun +
5         1;
6     }
7 }
8 System.out.println("Matrisin Icerigi:");
9 for (int satir = 0; satir < 3; satir++) {
10     for (int sutun = 0; sutun < 3; sutun++) {
11         System.out.print(matris[satir][sutun] + "
12 ");
13     }
14     System.out.println();
15 }
```

	0	1	2
0	1	2	3
1	4	0	0
2	0	0	0

$\text{matris}[1][0] = 1 * 3 + 0 + 1 = 4$. İç döngü bitti; dış döngü bir sonraki **satıra** geçti.

Adım 5 / 11

DEĞİŞKENLER

satir = 1

sutun = 0

ÇIKTI

```
Matrisin Icerigi:
1 2 3
4 0 0
0 0 0
```

Matris Doldurma ve Yazdırma: Adım Adım

```
1 int[][] matris = new int[3][3];
2 for (int satir = 0; satir < 3; satir++) {
3     for (int sutun = 0; sutun < 3; sutun++) {
4         matris[satir][sutun] = satir * 3 + sutun +
5         1;
6     }
7 }
8 System.out.println("Matrisin Icerigi:");
9 for (int satir = 0; satir < 3; satir++) {
10     for (int sutun = 0; sutun < 3; sutun++) {
11         System.out.print(matris[satir][sutun] + "
12 ");
13     }
14     System.out.println();
15 }
```

	0	1	2
0	1	2	3
1	4	5	0
2	0	0	0

`matris[1][1] = 1 * 3 + 1 + 1 = 5`

Adım 6 / 11

DEĞİŞKENLER

satir = 1

sutun = 1

ÇIKTI

Matris Doldurma ve Yazdırma: Adım Adım

```
1 int[][] matris = new int[3][3];
2 for (int satir = 0; satir < 3; satir++) {
3     for (int sutun = 0; sutun < 3; sutun++) {
4         matris[satir][sutun] = satir * 3 + sutun +
5         1;
6     }
7 }
8 System.out.println("Matrisin Icerigi:");
9 for (int satir = 0; satir < 3; satir++) {
10     for (int sutun = 0; sutun < 3; sutun++) {
11         System.out.print(matris[satir][sutun] + "
12 ");
13     }
14     System.out.println();
15 }
```

	0	1	2
0	1	2	3
1	4	5	6
2	0	0	0

`matris[1][2] = 1 * 3 + 2 + 1 = 6`

Adım 7 / 11

DEĞİŞKENLER

satir = 1

sutun = 2

ÇIKTI

Matris Doldurma ve Yazdırma: Adım Adım

```
1 int[][] matris = new int[3][3];
2 for (int satir = 0; satir < 3; satir++) {
3     for (int sutun = 0; sutun < 3; sutun++) {
4         matris[satir][sutun] = satir * 3 + sutun +
5         1;
6     }
7 }
8 System.out.println("Matrisin Icerigi:");
9 for (int satir = 0; satir < 3; satir++) {
10     for (int sutun = 0; sutun < 3; sutun++) {
11         System.out.print(matris[satir][sutun] + "
12 ");
13     }
14     System.out.println();
15 }
```

	0	1	2
0	1	2	3
1	4	5	6
2	7	0	0

$\text{matris}[2][0] = 2 * 3 + 0 + 1 = 7$. İç döngü bitti; dış döngü bir sonraki **satıra** geçti.

Adım 8 / 11

DEĞİŞKENLER

satir = 2

sutun = 0

ÇIKTI

```
Matrisin Icerigi:
1 2 3
4 5 6
7 0 0
```

Matris Doldurma ve Yazdırma: Adım Adım

```
1 int[][] matris = new int[3][3];
2 for (int satir = 0; satir < 3; satir++) {
3     for (int sutun = 0; sutun < 3; sutun++) {
4         matris[satir][sutun] = satir * 3 + sutun +
5         1;
6     }
7 }
8 System.out.println("Matrisin Icerigi:");
9 for (int satir = 0; satir < 3; satir++) {
10     for (int sutun = 0; sutun < 3; sutun++) {
11         System.out.print(matris[satir][sutun] + "
12 ");
13     }
14     System.out.println();
15 }
```

	0	1	2
0	1	2	3
1	4	5	6
2	7	8	0

`matris[2][1] = 2 * 3 + 1 + 1 = 8`

Adım 9 / 11

DEĞİŞKENLER

satir = 2

sutun = 1

ÇIKTI

Matris Doldurma ve Yazdırma: Adım Adım

```
1 int[][] matris = new int[3][3];
2 for (int satir = 0; satir < 3; satir++) {
3     for (int sutun = 0; sutun < 3; sutun++) {
4         matris[satir][sutun] = satir * 3 + sutun +
5         1;
6     }
7 }
8 System.out.println("Matrisin Icerigi:");
9 for (int satir = 0; satir < 3; satir++) {
10     for (int sutun = 0; sutun < 3; sutun++) {
11         System.out.print(matris[satir][sutun] + "
12 ");
13     }
14 }
15 System.out.println();
```

	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9

`matris[2][2] = 2 * 3 + 2 + 1 = 9`

Adım 10 / 11

DEĞİŞKENLER

satir = 2

sutun = 2

ÇIKTI

Matris Doldurma ve Yazdırma: Adım Adım

```
1 int[][] matris = new int[3][3];
2 for (int satir = 0; satir < 3; satir++) {
3     for (int sutun = 0; sutun < 3; sutun++) {
4         matris[satir][sutun] = satir * 3 + sutun +
5         1;
6     }
7 }
8 System.out.println("Matrisin Icerigi:");
9 for (int satir = 0; satir < 3; satir++) {
10     for (int sutun = 0; sutun < 3; sutun++) {
11         System.out.print(matris[satir][sutun] + "
12         ");
13     }
14     System.out.println();
15 }
```

	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9

Yazdırma döngüsü aynı sırayla dolaşır; her satırın sonunda `println()` alt satıra geçer.

Adım 11 / 11

DEĞİŞKENLER

—

ÇIKTI

Matrisin Icerigi:

```
1 2 3
4 5 6
7 8 9
```

ÇOK BOYUTLU DİZİLER

Üç ve Daha Fazla Boyut

- Java'da üç boyutlu ve daha fazla boyutlu diziler de kullanılabilir: her ek boyut bir köşeli parantez daha demektir.
- Daha fazla boyut daha karmaşık veri yapılarına izin verir: bina `[kat][satır][sutun]`, renkli resim `[y][x][kanal]`, zamanla değişen ölçüm tablosu.
- Öğe sayısı boyutların **çarpımıdır**: `new int[3][3][3]` → $3 \times 3 \times 3 = 27$ hücre. Boyut sayısı arttıkça bellek hızla büyür.

```
// 3x3x3 boyutunda üç boyutlu dizi
int[][][] ucBoyutluDizi = new int[3][3][3];
ucBoyutluDizi[1][2][0] = 16; // katman 1, satır 2, sütun 0
```

ÜÇ BOYUTLU DİZİ: 3 KATMAN, HER KATMAN 3×3 BİR MATRİS

	x = 0		x = 1		x = 2
0	123	0	101112	0	192021
1	456	1	131415	1	222324
2	789	2	161718	2	252627

$$\text{ucBoyutluDizi}[1][2][0] = 1 \cdot 9 + 2 \cdot 3 + 0 + 1 = 16$$

ÇOK BOYUTLU DİZİLER

Örnek: UcBoyutluDiziOrnegi

```
int[][][] ucBoyutluDizi = new int[3][3][3];
for (int x = 0; x < 3; x++) {
    for (int y = 0; y < 3; y++) {
        for (int z = 0; z < 3; z++) {
            ucBoyutluDizi[x][y][z] = x * 9 + y * 3 + z + 1;
        }
    }
}

System.out.println("Uc Boyutlu Dizinin Icerigi:");
for (int x = 0; x < 3; x++) {
    for (int y = 0; y < 3; y++) {
        for (int z = 0; z < 3; z++) {
            System.out.print(ucBoyutluDizi[x][y][z] + " ");
        }
        System.out.println();
    }
}
System.out.println();
}
```

ÇIKTI

Uc Boyutlu Dizinin Icerigi:

1 2 3
4 5 6
7 8 9

10 11 12
13 14 15
16 17 18

19 20 21
22 23 24
25 26 27

Üç iç içe döngü: **z** bir satırı,
y bir katmanı, **x** tüm diziyi
dolaşır. Her katmandan sonra
boş satır basılır.

Matris Transpozu

```
public static void main(String[] args) {  
    int[][] matris = {  
        {1, 2, 3},  
        {4, 5, 6}  
    };  
    System.out.println("Orijinal Matris:");  
    yazdirMatris(matris);  
    int satirSayisi = matris.length;  
    int sutunSayisi = matris[0].length;  
    int[][] transpozMatris = new int[sutunSayisi][satirSayisi];  
    for (int i = 0; i < satirSayisi; i++) {  
        for (int j = 0; j < sutunSayisi; j++) {  
            transpozMatris[j][i] = matris[i][j];  
        }  
    }  
    System.out.println("Transpoz Matris:");  
    yazdirMatris(transpozMatris);  
}
```

ÇIKTI

Orijinal Matris:

1 2 3

4 5 6

Transpoz Matris:

1 4

2 5

3 6

- Transpoz, matrisin **satırlarını sütunlara**, sütunlarını satırlara dönüştürür: `T[j][i] = M[i][j]`.
- 2×3 matrisin transpozu 3×2'dir: `new int[sutunSayisi][satirSayisi]`.

Transpoz: Hücre Hücre Kopyalama

matris (2×3)

	0	1	2
0	1	2	3
1	4	5	6

transpozMatris (3×2)

	0	1
0	0	0
1	0	0
2	0	0

`satirSayisi = 2` , `sutunSayisi = 3` ; transpoz için 3×2'lik yeni bir dizi oluşturulur (hücreler 0).

Adım 1 / 7

```
for (int i = 0; i < satirSayisi; i++) {  
    for (int j = 0; j < sutunSayisi; j++) {  
        transpozMatris[j][i] = matris[i][j];  
    }  
}
```

DEĞİŞKENLER

`i = -`

`j = -`

Transpoz: Hücre Hücre Kopyalama

matris (2×3)

	0	1	2
0	1	2	3
1	4	5	6

transpozMatris (3×2)

	0	1
0	1	0
1	0	0
2	0	0

`transpozMatris[0][0] = matris[0][0]` = 1 · matrisin **satırı**, transpozun **sütunu** oluyor.

Adım 2 / 7

```
for (int i = 0; i < satirSayisi; i++) {  
    for (int j = 0; j < sutunSayisi; j++) {  
        transpozMatris[j][i] = matris[i][j];  
    }  
}
```

DEĞİŞKENLER

`i = 0`

`j = 0`

Transpoz: Hücre Hücre Kopyalama

matris (2×3)

	0	1	2
0	1	2	3
1	4	5	6

transpozMatris (3×2)

	0	1
0	1	0
1	2	0
2	0	0

```
transpozMatris[1][0] = matris[0][1] = 2
```

Adım 3 / 7

```
for (int i = 0; i < satirSayisi; i++) {  
    for (int j = 0; j < sutunSayisi; j++) {  
        transpozMatris[j][i] = matris[i][j];  
    }  
}
```

DEĞİŞKENLER

i = 0

j = 1

Transpoz: Hücre Hücre Kopyalama

matris (2×3)

	0	1	2
0	1	2	3
1	4	5	6

transpozMatris (3×2)

	0	1
0	1	0
1	2	0
2	3	0

```
transpozMatris[2][0] = matris[0][2] = 3
```

Adım 4 / 7

```
for (int i = 0; i < satirSayisi; i++) {  
    for (int j = 0; j < sutunSayisi; j++) {  
        transpozMatris[j][i] = matris[i][j];  
    }  
}
```

DEĞİŞKENLER

i = 0

j = 2

Transpoz: Hücre Hücre Kopyalama

matris (2×3)

	0	1	2
0	1	2	3
1	4	5	6

transpozMatris (3×2)

	0	1
0	1	4
1	2	0
2	3	0

`transpozMatris[0][1] = matris[1][0]` = 4 · matrisin **satırı**, transpozun **sütunu** oluyor.

Adım 5 / 7

```
for (int i = 0; i < satirSayisi; i++) {  
    for (int j = 0; j < sutunSayisi; j++) {  
        transpozMatris[j][i] = matris[i][j];  
    }  
}
```

DEĞİŞKENLER

`i = 1`

`j = 0`

Transpoz: Hücre Hücre Kopyalama

matris (2×3)

	0	1	2
0	1	2	3
1	4	5	6

transpozMatris (3×2)

	0	1
0	1	4
1	2	5
2	3	0

```
transpozMatris[1][1] = matris[1][1] = 5
```

Adım 6 / 7

```
for (int i = 0; i < satirSayisi; i++) {  
    for (int j = 0; j < sutunSayisi; j++) {  
        transpozMatris[j][i] = matris[i][j];  
    }  
}
```

DEĞİŞKENLER

i = 1

j = 1

Transpoz: Hücre Hücre Kopyalama

matris (2×3)

	0	1	2
0	1	2	3
1	4	5	6

transpozMatris (3×2)

	0	1
0	1	4
1	2	5
2	3	6

```
transpozMatris[2][1] = matris[1][2] = 6
```

Adım 7 / 7

```
for (int i = 0; i < satirSayisi; i++) {  
    for (int j = 0; j < sutunSayisi; j++) {  
        transpozMatris[j][i] = matris[i][j];  
    }  
}
```

DEĞİŞKENLER

i = 1

j = 2

Yardımcı Metot: `yazdirMatris`

```
public static void yazdirMatris(int[][] matris) {  
    for (int i = 0; i < matris.length; i++) {  
        for (int j = 0; j < matris[i].length; j++) {  
            System.out.print(matris[i][j] + " ");  
        }  
        System.out.println();  
    }  
}
```

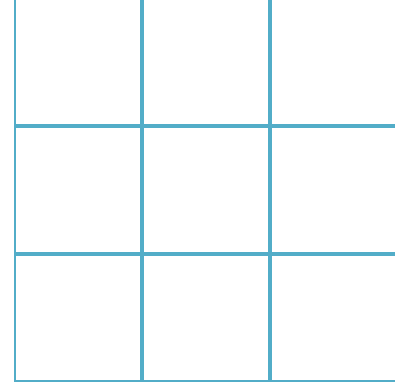
- Yazdırma işi bir **yardımcı metoda** taşınmıştır; `main` iki kez çağırır: önce asıl matris, sonra transpoz.
- Parametre `int[][] matris` : her boyuttaki 2B diziyle çalışır, çünkü satır sayısı `matris.length` , sütun sayısı `matris[i].length` ile **diziden okunur**.
- Metoda dizinin kopyası değil **referansı** gider; büyük bir matris için bile çağrı ucuzdur.
- Sabit `3` yazılıysaydı 2×3 ve 3×2 matrislerden biri hatalı yazdırılır ya da `ArrayIndexOutOfBoundsException` oluşurdu.

UYGULAMA: TIC TAC TOE

Tic Tac Toe (X-O Oyunu)

- 3×3'lük bir oyun tahtası kullanılır:
`char[][] tahta = new char[3][3];`
- İki oyuncu (X ve O) sırayla hamle yapar; her hamle tek bir atamadır:
`tahta[satir][sutun] = oyuncu .`
- Kendi işaretinden **3 tanesini** aynı satıra, sütuna ya da çapraza yerleştiren oyunu kazanır.
- Boş hücre kalmazsa oyun berabere biter.

Sağdaki tahtada oynayın: her tıklamada çalışan dizi ataması gösterilir.



Sıra **X** oyuncusunda. Bir hücreye tıkla.

yenı oyun

UYGULAMA: TIC TAC TOE

TicTacToe: main

```
import java.util.Scanner;

public class TicTacToe {
    static Scanner scanner = new Scanner(System.in);

    public static void main(String[] args) {
        char[][] tahta = new char[3][3]; // 3x3'lük tahta
        char oyuncu = 'X'; // İlk oyuncu X
        boolean oyunDevamEdiyor = true;
        tahtaDoldur(tahta); // Tüm hücreler boş
        while (oyunDevamEdiyor) {
            tahtayiGoster(tahta);
            hamleYap(tahta, oyuncu);
            oyunDevamEdiyor = oyunDevamEdiyorMu(tahta, oyuncu);
            // Oyuncu değişimi
            oyuncu = (oyuncu == 'X') ? 'O' : 'X';
        }
        tahtayiGoster(tahta);
    }
    // ... metotlar
}
```

- İş **metotlara** bölünmüştür: doldur, göster, hamle al, kontrol et.
- Tahta metotlara **referansla** gider: `hamleYap` 'ın yaptığı değişiklik `main` 'de görünür.
- Metotlar `static` olmalıdır; değilse:

error: non-static method
tahtaDoldur(char[][][]) cannot
be referenced from a static
context

TicTacToe: Tahtayı Doldur ve Göster

```
public static void tahtaDoldur(char[][] tahta) {
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            tahta[i][j] = ' ';
        }
    }
}

public static void tahtayiGoster(char[][] tahta) {
    System.out.println("-----");
    for (int i = 0; i < 3; i++) {
        System.out.print("| ");
        for (int j = 0; j < 3; j++) {
            System.out.print(tahta[i][j] + " | ");
        }
        System.out.println("\n-----");
    }
}
```

BOŞ TAHTANIN GÖRÜNÜMÜ

```
-----
| | | |
-----
| | | |
-----
| | | |
-----
```

- `new char[3][3]` hücreleri `'\u0000'` ile başlar; ekranda düzgün görünmesi ve boş hücre kontrolü için önce `' '` (boşluk) ile doldurulur.
- `"\n-----"` : satır sonu ve ayırıcı çizgi tek `println` ile.

TicTacToe: Hamle Alma

```
static Scanner scanner = new Scanner(System.in);

public static void hamleYap(char[][] tahta, char oyuncu) {
    int satir, sutun;
    do {
        System.out.print("Sıra " + oyuncu
            + " oyuncusunda. Satır ve sütun seçin (1-3): ");
        satir = scanner.nextInt() - 1;
        sutun = scanner.nextInt() - 1;
    } while (satir < 0 || satir > 2 || sutun < 0 || sutun > 2
        || tahta[satir][sutun] != ' ');
    tahta[satir][sutun] = oyuncu;
}
```

0 tabanlı indis

Kullanıcı 1–3 arası girer; `- 1` ile 0–2 indisine çevrilir.

do-while

Geçersiz ya da **dolu** hücre seçildikçe soru yeniden sorulur; geçerli seçimde döngü biter.

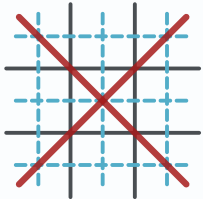
Kısa devre

Sınır kontrolleri `tahta[satir][sutun]` 'dan **önce** yazılır: `||` ilk doğru koşulda durduğu için sınır dışı erişim olmaz.

UYGULAMA: TIC TAC TOE

TicTacToe: Kazanan Var mı?

```
public static boolean oyunDevamEdiyorMu(char[][] tahta, char oyuncu) {  
    // Kazanan durumlarını kontrol et  
    if ((tahta[0][0] == oyuncu && tahta[0][1] == oyuncu && tahta[0][2] == oyuncu) ||  
        (tahta[1][0] == oyuncu && tahta[1][1] == oyuncu && tahta[1][2] == oyuncu) ||  
        (tahta[2][0] == oyuncu && tahta[2][1] == oyuncu && tahta[2][2] == oyuncu) ||  
        (tahta[0][0] == oyuncu && tahta[1][0] == oyuncu && tahta[2][0] == oyuncu) ||  
        (tahta[0][1] == oyuncu && tahta[1][1] == oyuncu && tahta[2][1] == oyuncu) ||  
        (tahta[0][2] == oyuncu && tahta[1][2] == oyuncu && tahta[2][2] == oyuncu) ||  
        (tahta[0][0] == oyuncu && tahta[1][1] == oyuncu && tahta[2][2] == oyuncu) ||  
        (tahta[0][2] == oyuncu && tahta[1][1] == oyuncu && tahta[2][0] == oyuncu)) {  
        System.out.println("Tebrikler, " + oyuncu + " oyuncusu kazandı!");  
        return false;  
    }  
    // ... berabere kontrolü
```



3 satır + 3 sütun + 2 çapraz

- 8 kazanma çizgisi tek tek denetlenir: her biri **3 hücrenin** de `oyuncu` olmasını ister (`&&`), çizgilerden **biri** yeterlidir (`||`).
- Yalnızca son hamleyi yapan oyuncu denetlenir: yeni bir çizgiyi ancak o tamamlamış olabilir.
- Kazanan varsa metot `false` döndürür: "oyun devam etmiyor".

UYGULAMA: TIC TAC TOE

TicTacToe: Berabere mi?

```
// oyunDevamEdiyorMu metodunun devamı
// Berabere durumunu kontrol et
boolean berabere = true;
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 3; j++) {
        if (tahta[i][j] == ' ') {
            berabere = false;
            break;
        }
    }
    if (!berabere) {
        break;
    }
}
if (berabere) {
    System.out.println("Oyun berabere bitti.");
    return false;
}
return true;
```

- Önce oyunun berabere olduğu **varsayılır** (`berabere = true`); tek bir boş hücre bu varsayımı çürütür.
- İç `break` yalnızca **iç** döngüden çıkar; dıştaki `if (!berabere) break;` dış döngüyü de bitirir.
- Kazanan yoksa ve boş hücre varsa `true` : oyun devam eder.
- Önce kazanma, sonra beraberlik denetlenir: son hamle tahtayı doldurup aynı anda kazandırabilir.

Beraberlik: İki `break`, İki Döngü

Tahta: `X O X` · `O X _` · `O X O` (·: boşluk karakteri)

```
▶ boolean berabere = true;
2 for (int i = 0; i < 3; i++) {
3     for (int j = 0; j < 3; j++) {
4         if (tahta[i][j] == ' ') {
5             berabere = false;
6             break;
7         }
8     }
9     if (!berabere) {
10        break;
11    }
12 }
13 System.out.println(berabere ? "berabere" : "devam");
```

	0	1	2	
0	X	O	X	berabere = true
1	O	X	_	
2	O	X	O	

Önce beraberlik varsayılır. Tahtada tek boş hücre var: (1, 2).

Adım 1 / 13

DEĞİŞKENLER

berabere = **true**

ÇIKTI

Beraberlik: İki `break`, İki Döngü

Tahta: `X O X` · `O X _` · `O X O` (·: boşluk karakteri)

```
1 boolean berabere = true;
2 for (int i = 0; i < 3; i++) {
3     for (int j = 0; j < 3; j++) {
4         if (tahta[i][j] == ' ') {
5             berabere = false;
6             break;
7         }
8     }
9     if (!berabere) {
10        break;
11    }
12 }
13 System.out.println(berabere ? "berabere" : "devam");
```

	0	1	2	
0	X	O	X	berabere = true i = 0, j = 0
1	O	X	_	
2	O	X	O	

(0, 0) = 'X': boş değil, devam.

Adım 2 / 13

DEĞİŞKENLER

berabere = true

i = 0

j = 0

ÇIKTI

Beraberlik: İki `break`, İki Döngü

Tahta: `X O X` · `O X _` · `O X O` (`_`: boşluk karakteri)

```
1 boolean berabere = true;
2 for (int i = 0; i < 3; i++) {
3     for (int j = 0; j < 3; j++) {
4         if (tahta[i][j] == ' ') {
5             berabere = false;
6             break;
7         }
8     }
9     if (!berabere) {
10        break;
11    }
12 }
13 System.out.println(berabere ? "berabere" : "devam");
```

	0	1	2
0	X	O	X
1	O	X	_
2	O	X	O

(0, 1) = 'O': boş değil.

berabere =

true

i = 0, j = 1

Adım 3 / 13

DEĞİŞKENLER

berabere = **true**

i = **0**

j = **1**

ÇIKTI

Beraberlik: İki `break`, İki Döngü

Tahta: `X O X` · `O X _` · `O X O` (·: boşluk karakteri)

```
1 boolean berabere = true;
2 for (int i = 0; i < 3; i++) {
3     for (int j = 0; j < 3; j++) {
4         if (tahta[i][j] == 'X') {
5             berabere = false;
6             break;
7         }
8     }
9     if (!berabere) {
10        break;
11    }
12 }
13 System.out.println(berabere ? "berabere" : "devam");
```

	0	1	2
0	X	O	X
1	O	X	_
2	O	X	O

berabere = **true**
i = 0, j = 2

(0, 2) = 'X': satır 0 dolu; iç döngü kendiliğinden biter.

Adım 4 / 13

DEĞİŞKENLER

berabere = **true**

i = **0**

j = **2**

ÇIKTI

Beraberlik: İki `break`, İki Döngü

Tahta: `X O X` · `O X _` · `O X O` (·: boşluk karakteri)

```
1 boolean berabere = true;
2 for (int i = 0; i < 3; i++) {
3     for (int j = 0; j < 3; j++) {
4         if (tahta[i][j] == ' ') {
5             berabere = false;
6             break;
7         }
8     }
9     if (!berabere) {
10        break;
11    }
12 }
13 System.out.println(berabere ? "berabere" : "devam");
```

	0	1	2
0	X	O	X
1	O	X	_
2	O	X	O

berabere = **true**
i = 0

`!berabere` yanlış: dış döngü sürer.

Adım 5 / 13

DEĞİŞKENLER

berabere = **true**

i = 0

ÇIKTI

Beraberlik: İki `break`, İki Döngü

Tahta: `X O X` · `O X _` · `O X O` (·: boşluk karakteri)

```
1 boolean berabere = true;
2 for (int i = 0; i < 3; i++) {
3     for (int j = 0; j < 3; j++) {
4         if (tahta[i][j] == ' ') {
5             berabere = false;
6             break;
7         }
8     }
9     if (!berabere) {
10        break;
11    }
12 }
13 System.out.println(berabere ? "berabere" : "devam");
```

	0	1	2
0	X	O	X
1	O	X	_
2	O	X	O

i = 1: (1, 0) = 'O'.

berabere =

true

i = 1, j = 0

Adım 6 / 13

DEĞİŞKENLER

berabere = **true**

i = 1

j = 0

ÇIKTI

Beraberlik: İki `break`, İki Döngü

Tahta: `X O X` · `O X _` · `O X O` (·: boşluk karakteri)

```
1 boolean berabere = true;
2 for (int i = 0; i < 3; i++) {
3     for (int j = 0; j < 3; j++) {
4         if (tahta[i][j] == 'X') {
5             berabere = false;
6             break;
7         }
8     }
9     if (!berabere) {
10        break;
11    }
12 }
13 System.out.println(berabere ? "berabere" : "devam");
```

	0	1	2
0	X	O	X
1	O	X	_
2	O	X	O

(1, 1) = 'X'.

berabere =
true

i = 1, j = 1

Adım 7 / 13

DEĞİŞKENLER

berabere = **true**

i = **1**

j = **1**

ÇIKTI

Beraberlik: İki `break`, İki Döngü

Tahta: `X O X` · `O X _` · `O X O` (·: boşluk karakteri)

```
1 boolean berabere = true;
2 for (int i = 0; i < 3; i++) {
3     for (int j = 0; j < 3; j++) {
4         if (tahta[i][j] == ' ') {
5             berabere = false;
6             break;
7         }
8     }
9     if (!berabere) {
10        break;
11    }
12 }
13 System.out.println(berabere ? "berabere" : "devam");
```

	0	1	2	
0	X	O	X	
1	O	X	_	berabere = true i = 1, j = 2
2	O	X	O	

(1, 2) = ' ': boş hücre bulundu.

Adım 8 / 13

DEĞİŞKENLER

berabere = **true**

i = 1

j = 2

ÇIKTI

Beraberlik: İki `break`, İki Döngü

Tahta: `X O X` · `O X _` · `O X O` (·: boşluk karakteri)

```
1 boolean berabere = true;
2 for (int i = 0; i < 3; i++) {
3     for (int j = 0; j < 3; j++) {
4         if (tahta[i][j] == ' ') {
5             ▶ berabere = false;
6             break;
7         }
8     }
9     if (!berabere) {
10        break;
11    }
12 }
13 System.out.println(berabere ? "berabere" : "devam");
```

	0	1	2
0	X	O	X
1	O	X	_
2	O	X	O

berabere =

false

i = 1, j = 2

Varsayım çürüdü: `berabere = false` .

Adım 9 / 13

DEĞİŞKENLER

`berabere = false`

`i = 1`

`j = 2`

ÇIKTI

Beraberlik: İki `break`, İki Döngü

Tahta: `X O X` · `O X _` · `O X O` (·: boşluk karakteri)

```
1 boolean berabere = true;
2 for (int i = 0; i < 3; i++) {
3     for (int j = 0; j < 3; j++) {
4         if (tahta[i][j] == ' ') {
5             berabere = false;
6             break;
7         }
8     }
9     if (!berabere) {
10        break;
11    }
12 }
13 System.out.println(berabere ? "berabere" : "devam");
```

	0	1	2
0	X	O	X
1	O	X	_
2	O	X	O

berabere =

false

i = 1, j = 2

İç `break` : yalnızca iç döngüden çıkar; dış döngü hâlâ çalışıyor.

Adım 10 / 13

DEĞİŞKENLER

berabere = **false**

i = **1**

j = **2**

ÇIKTI

Beraberlik: İki `break`, İki Döngü

Tahta: `X O X` · `O X _` · `O X O` (·: boşluk karakteri)

```
1 boolean berabere = true;
2 for (int i = 0; i < 3; i++) {
3     for (int j = 0; j < 3; j++) {
4         if (tahta[i][j] == ' ') {
5             berabere = false;
6             break;
7         }
8     }
9     if (!berabere) {
10        break;
11    }
12 }
13 System.out.println(berabere ? "berabere" : "devam");
```

	0	1	2
0	X	O	X
1	O	X	_
2	O	X	O

`!berabere` doğru.

berabere =
false
i = 1

Adım 11 / 13

DEĞİŞKENLER

berabere = **false**

i = **1**

ÇIKTI

Beraberlik: İki `break`, İki Döngü

Tahta: `X O X` · `O X _` · `O X O` (·: boşluk karakteri)

```
1 boolean berabere = true;
2 for (int i = 0; i < 3; i++) {
3     for (int j = 0; j < 3; j++) {
4         if (tahta[i][j] == ' ') {
5             berabere = false;
6             break;
7         }
8     }
9     if (!berabere) {
10        break;
11    }
12 }
13 System.out.println(berabere ? "berabere" : "devam");
```

	0	1	2
0	X	O	X
1	O	X	_
2	O	X	O

berabere =

false

i = 1

Dış `break` : satır 2 hiç denetlenmez.

Adım 12 / 13

DEĞİŞKENLER

berabere = **false**

i = **1**

ÇIKTI

Beraberlik: İki `break`, İki Döngü

Tahta: `X O X` · `O X _` · `O X O` (·: boşluk karakteri)

```
1 boolean berabere = true;
2 for (int i = 0; i < 3; i++) {
3     for (int j = 0; j < 3; j++) {
4         if (tahta[i][j] == ' ') {
5             berabere = false;
6             break;
7         }
8     }
9     if (!berabere) {
10        break;
11    }
12 }
```

```
► System.out.println(berabere ? "berabere" : "devam");
```

	0	1	2
0	X	O	X
1	O	X	_
2	O	X	O

berabere =
false

Boş hücre var: `devam` yazılır.

Adım 13 / 13

DEĞİŞKENLER

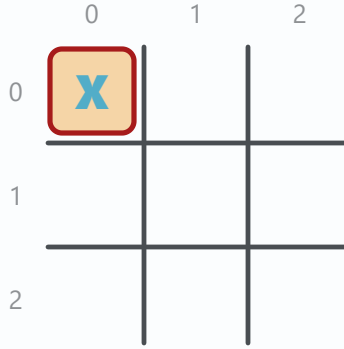
berabere = **false**

ÇIKTI

devam

UYGULAMA: TIC TAC TOE

Örnek Oyun



X 1 1 girer → `tahta[0][0] = 'X'` .
Kazanan yok, boş hücre var → `true` ; sıra
O'ye geçer.

Adım 1 / 6

GİRDİLER (SATIR SÜTUN)

1 1 1 2 2 2 2 2 1 3 3 3

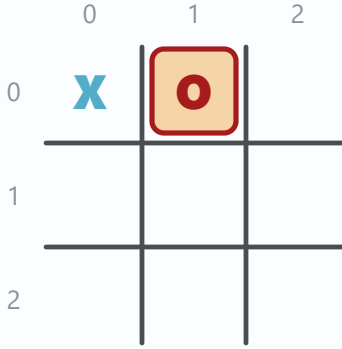
PROGRAMIN SON ÇIKTISI

Tebrikler, X oyuncusu kazandı!

```
-----  
| x | o | o |  
-----  
|   | x |   |  
-----  
|   |   | x |  
-----
```

UYGULAMA: TIC TAC TOE

Örnek Oyun



O **1 2** girer → `tahta[0][1] = 'O'` .
Kazanan yok, boş hücre var → `true` ; sıra
X'ye geçer.

Adım 2 / 6

GİRDİLER (SATIR SÜTUN)

1 1 1 2 2 2 2 2 1 3 3 3

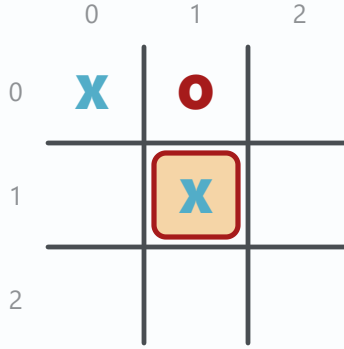
PROGRAMIN SON ÇIKTISI

Tebrikler, X oyuncusu kazandı!

```
-----  
| x | o | o |  
-----  
|   | x |   |  
-----  
|   |   | x |  
-----
```

UYGULAMA: TIC TAC TOE

Örnek Oyun



X 2 2 girer → `tahta[1][1] = 'X'` .
Kazanan yok, boş hücre var → `true` ; sıra
O'ye geçer.

Adım 3 / 6

GİRDİLER (SATIR SÜTUN)

1 1 1 2 2 2 2 2 1 3 3 3

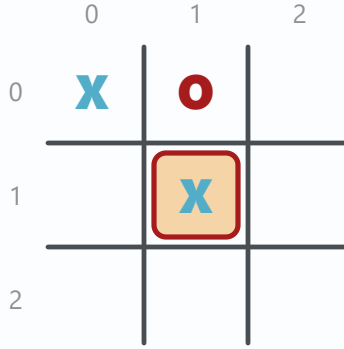
PROGRAMIN SON ÇIKTISI

Tebrikler, X oyuncusu kazandı!

```
-----  
| x | o | o |  
-----  
|   | x |   |  
-----  
|   |   | x |  
-----
```

UYGULAMA: TIC TAC TOE

Örnek Oyun



GİRDİLER (SATIR SÜTUN)

1 1 1 2 2 2 2 2 1 3 3 3

PROGRAMIN SON ÇIKTISI

Tebrikler, X oyuncusu kazandı!

```
-----  
| x | o | o |  
-----  
|   | x |   |  
-----  
|   |   | x |  
-----
```

O 2 2 girer: `tahta[1][1]` dolu (`!= ' '`) → `do-while` koşulu doğru, aynı soru tekrar sorulur.

UYGULAMA: TIC TAC TOE

Örnek Oyun

	0	1	2
0	X	O	O
1		X	
2			

O 1 3 girer → `tahta[0][2] = 'O'` .
Kazanan yok, boş hücre var → `true` ; sıra
X'ye geçer.

Adım 5 / 6

GİRDİLER (SATIR SÜTUN)

1 1 1 2 2 2 2 2 1 3 3 3

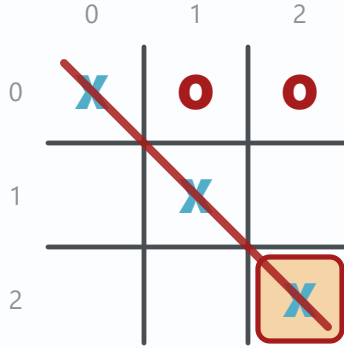
PROGRAMIN SON ÇIKTISI

Tebrikler, X oyuncusu kazandı!

```
-----  
| x | o | o |  
-----  
|   | x |   |  
-----  
|   |   | x |  
-----
```

UYGULAMA: TIC TAC TOE

Örnek Oyun



X 3 3 girer → `tahta[2][2] = 'X'` . Ana
çapraz X → kazanma mesajı,
`oyunDevamEdiyorMu` `false` döner,
döngü biter.

GİRDİLER (SATIR SÜTUN)

1 1 1 2 2 2 2 2 1 3 3 3

PROGRAMIN SON ÇIKTISI

Tebrikler, X oyuncusu kazandı!

```
-----  
| x | o | o |  
-----  
|   | x |   |  
-----  
|   |   | x |  
-----
```

ÖZET

Özet

Konu	Özü
Dizi	Aynı tipten, sabit sayıda öge; indis 0 ... length - 1
Oluşturma	<code>new int[n]</code> (varsayılan değerler) · <code>{1, 2, 3}</code> (yalnızca tanımlamada) · <code>new int[]{...}</code>
length	Özellik, parantezsiz: <code>dizi.length</code> ; 2B'de <code>m.length</code> satır, <code>m[i].length</code> sütun
Referans	Değişken adres tutar; <code>b = a</code> aynı dizi, <code>clone()</code> / <code>Arrays.copyOf</code> yeni dizi; <code>==</code> adres karşılaştırır
Bellek	Tüm diziler heap'te; referans yığında; erişilemeyen diziyi çöp toplayıcı temizler
Hata	Sınır dışı indis → <code>ArrayIndexOutOfBoundsException</code> ; döngüde <code>i < length</code>
Algoritmalar	Erişim $O(1)$ · doğrusal arama, ters çevirme $O(n)$ · eşleştirme, seçimli sıralama $O(n^2)$
Çok boyut	Dizilerin dizisi; iç içe döngüler; satırlar farklı uzunlukta olabilir

ÖZET

Sık Yapılan Hatalar (1)

Sonradan kısa ilk değer verme

```
a = {1, 2, 3};  
error: illegal start of expression → a =  
new int[]{1, 2, 3};
```

length() yazmak

```
a.length()  
error: cannot find symbol ... method  
length() → a.length
```

Boyutu unutmak

```
new int[]  
error: array dimension missing
```

Yanlış tipte öge

```
int[] a = {1, 2.5, 3};  
error: incompatible types: possible  
lossy conversion from double to int
```

Dalın içinde tanımlanan değişken

Ortanca örneğindeki gibi: error: cannot find
symbol ... variable ortanca

static olmayan metot

```
error: non-static method ... cannot be  
referenced from a static context
```

Sık Yapılan Hatalar (2)

Son indisi length sanmak

`a[a.length]` ya da `i <= a.length`:
`ArrayIndexOutOfBoundsException: Index 3 out of bounds for length 3`

Oluşturulmamış dizi

`int[] a = null; a[0]`:
`NullPointerException: Cannot load from int array because "a" is null`

Diziyi doğrudan yazdırmak

`System.out.println(a)` → `[I@7ef20235]`.
İçerik için `Arrays.toString(a)`.

Atamayı kopya sanmak

`b = a` sonrası `b[0]` 'ı değiştirmek `a[0]` 'ı da değiştirir. Kopya: `a.clone()`.

`==` ile içerik karşılaştırmak

Aynı öğeli iki dizi için `false`.
`Arrays.equals(a, b)` kullanın.

İlk indisi 1 sanmak

Kullanıcıya "1. öge" gösterilir ama kodda `dizi[0]` 'dır: `(i + 1) + ". eleman"`.

ÖZET

Örnek Problemler (1)

```
static int gizem(int[] d) {  
    int s = 0;  
    for (int i = 0; i < d.length; i += 2) {  
        s += d[i];  
    }  
    return s;  
}  
  
static void degistir(int[] d, int x) {  
    d[0] = 100;  
    x = 100;  
}  
  
// main:  
int[] d = {4, 8, 15, 16, 23, 42};  
int[] a = {1, 2, 3};  
int x = 1;  
degistir(a, x);
```

1. `gizem(d)` ne döndürür? Metot ne hesaplar?
2. `d[d.length / 2]` kaçtır?
3. `degistir(a, x)` sonrası `a[0]` ve `x` kaçtır? Neden farklılar?

► Cevapları göster

ÖZET

Örnek Problemler (2)

```
int[][] m = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
int iz = 0;
for (int i = 0; i < m.length; i++) {
    iz += m[i][i];
}
// ---
int[] b = new int[5];
for (int i = 1; i < b.length; i++) {
    b[i] = b[i - 1] + i;
}
System.out.println(Arrays.toString(b));
// ---
int[] p = {3, 1, 2};
int[] q = p;
q[2] = q[0] + q[1];
System.out.println(Arrays.toString(p));
```

4. Birinci parçadaki `iz` kaçır? Hangi hücreler toplanır?
5. İkinci parça ne yazdırır?
6. Üçüncü parça ne yazdırır?

► Cevapları göster

ÖZET

Alıştırmalar

1. `int enBuyuk(int[] dizi)` : en büyük öğeyi döndürün. `{12, 7, 30, 5, 18}` → ?
2. `int ciftSay(int[] dizi)` : çift öğelerin sayısını döndürün (for-each ile). Aynı dizi → ?
3. `int indisBul(int[] dizi, int hedef)` : hedefin indisini, yoksa -1 döndürün. 5 ve 99 için → ?
4. `int[][] topla(int[][] a, int[][] b)` : iki matrisin toplamı. `{{1, 2}, {3, 4}}` + `{{5, 6}, {7, 8}}` → ?
5. `int[] satirToplamlari(int[][] m)` : her satırın toplamı. `{{1, 2, 3}, {4, 5, 6}}` → ?
6. Tic Tac Toe'ya `tahtaDoluMu` metodu ekleyip beraberlik kontrolünü onunla yazın.

► Cevaplar / ipuçları

Kaynaklar

- Oracle, *The Java Tutorials*: Arrays; The for Statement (enhanced for).
- Oracle, Java SE 21 API: `java.util.Arrays` .
- J. Gosling ve diğ., *The Java Language Specification*, Java SE 21 Edition (Bölüm 10: Arrays).
- P. Deitel ve H. Deitel, *Java: How to Program*, 11. baskı, Pearson (Bölüm 7: Arrays and ArrayLists).
- K. Sierra ve B. Bates, *Head First Java*, 3. baskı, O'Reilly.
- C. S. Horstmann, *Core Java, Volume I: Fundamentals*, 12. baskı, Pearson (3.10 Arrays).
- A. Downey ve C. Mayfield, *Think Java*, 2. baskı, O'Reilly (Bölüm 7: Arrays and References).

Son

Bölüm 5: Diziler

Sonraki bölüm: Bölüm 5: Koleksiyonlar · [Terim sözlüğü](#)