

Bölüm 5: Koleksiyonlar

Java ile Nesne Yönelimli Programlama

Koleksiyon çerçevesi

Jenerik tipler

ArrayList ve LinkedList

Vector ve Stack

Queue ve PriorityQueue

Deque ve ArrayDeque

Set: HashSet, TreeSet

Map: HashMap, TreeMap

Iterator

Sıralama ve arama

Karıştırma ve kompozisyon

Sercan KÜLCÜ · Giresun Üniversitesi · Elektrik-Elektronik Mühendisliği

GİRİŞ

Öğrenme Çıktıları

Çerçeve

Koleksiyon çerçevesinin arayüz, uygulama ve algoritma unsurlarını ve arayüz/sınıf hiyerarşisini açıklamak.

Listeler

`ArrayList`, `LinkedList`, `Vector` ve `Stack` 'i kullanmak; performans farklarını açıklamak.

Küme ve eşlemeler

`HashSet`, `TreeSet`, `HashMap`, `TreeMap` ve bağlantılı sürümlerini doğru yerde seçmek.

Jenerik tipler

`List<String>` gibi jenerik tiplerle tip güvenli kod yazmak; sarmalayıcı sınıfları ve otomatik kutulamayı kullanmak.

Kuyruklar

`Queue`, `PriorityQueue` ve `ArrayDeque` ile FIFO, öncelik ve LIFO davranışlarını gerçekleştirmek.

Gezinme ve algoritmalar

`Iterator` ile güvenli gezinmek; `Collections` ile sıralamak, karıştırmak, aramak ve düzenlemek.

Neden Koleksiyonlar?

- **Koleksiyon (collection):** birden çok ögeyi tek bir nesnede saklayan ve yöneten veri yapısıdır.
- **Java Koleksiyon Çerçevesi** (Java Collections Framework): `java.util` paketindeki hazır arayüzler, sınıflar ve algoritmalar.
- Veri saklamak, sıralamak, aramak ve işlemek için güçlü, test edilmiş, performans ve güvenlik açısından optimize edilmiş araçlar sunar.
- Diziden farkı: boyut **dinamiktir**; `add`, `remove`, `contains` gibi hazır metotlar vardır.
- Doğru yapıyı seçmek önemlidir: her koleksiyon tipi farklı bir ihtiyaca göre tasarlanmıştır.

DİZİ: BOYUT BAŞTAN SABİT

```
int[] dizi = new int[3];    // boyut sabit: 3
dizi[0] = 10;
dizi[1] = 20;
dizi[2] = 30;
// dizi[3] = 40; → ArrayIndexOutOfBoundsException
```

LİSTE: GEREKTİKÇE BÜYÜR

```
List<Integer> liste = new ArrayList<>();
liste.add(10);
liste.add(20);
liste.add(30);
liste.add(40);    // sorun yok: büyür
System.out.println(liste);
System.out.println("boyut: " + liste.size());
```

```
[10, 20, 30, 40]
boyut: 4
```

Benzetme: dizi koltuk sayısı sabit bir otobüs;
`ArrayList` yolcu arttıkça büyüyen bir otobüstür.

Koleksiyon Türleri

List (liste)

Sıralı, indisli öğeler; tekrar olabilir.

`ArrayList` , `LinkedList`

Ali Can Ali Ece

Set (küme)

Yalnızca tekil öğeler; aynı öğe ikinci kez eklenmez. `HashSet` , `TreeSet`

Ali Can Ece

Map (eşleme)

Anahtar → değer çiftleri; her anahtar bir kez. `HashMap` , `TreeMap`

Ali → 85 Can → 72

Queue (kuyruk)

İşlenmeyi bekleyen öğeler; genellikle ilk giren ilk çıkar (FIFO). `LinkedList` ,

`PriorityQueue`

baş: Ali Can Ece: son

Temel İşlemler ve Tasarım Amaçları

Temel işlemler

Ekleme, çıkarma, güncelleme ve arama; `for-each` döngüsüyle kolay gezinme (iterasyon).

Kolay kullanım

Basit ve anlaşılır bir API: kod daha okunur olur, bakımı kolaylaşır.

Performans

Veri hacmine ve erişim ihtiyacına göre optimize edilmiş, bellek açısından verimli yapılar.

Tip güvenliği

Jenerik yapılar (`List<String>`) yanlış tipteki öğeyi derleme zamanında yakalar.

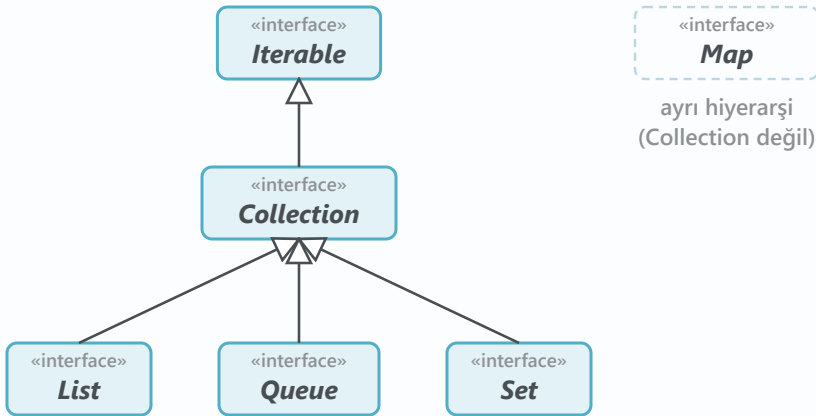
Geniş hiyerarşi

Farklı ihtiyaçlar için bir sınıf ailesi; standart kütüphanenin geri kalanıyla uyumlu çalışır.

Hazır algoritmalar

Sıralama, arama, karıştırma hazırdır; hatalı kullanım istisnaıyla bildirilir. Zaman kazandırır, güvenilir yazılıma katkı sağlar.

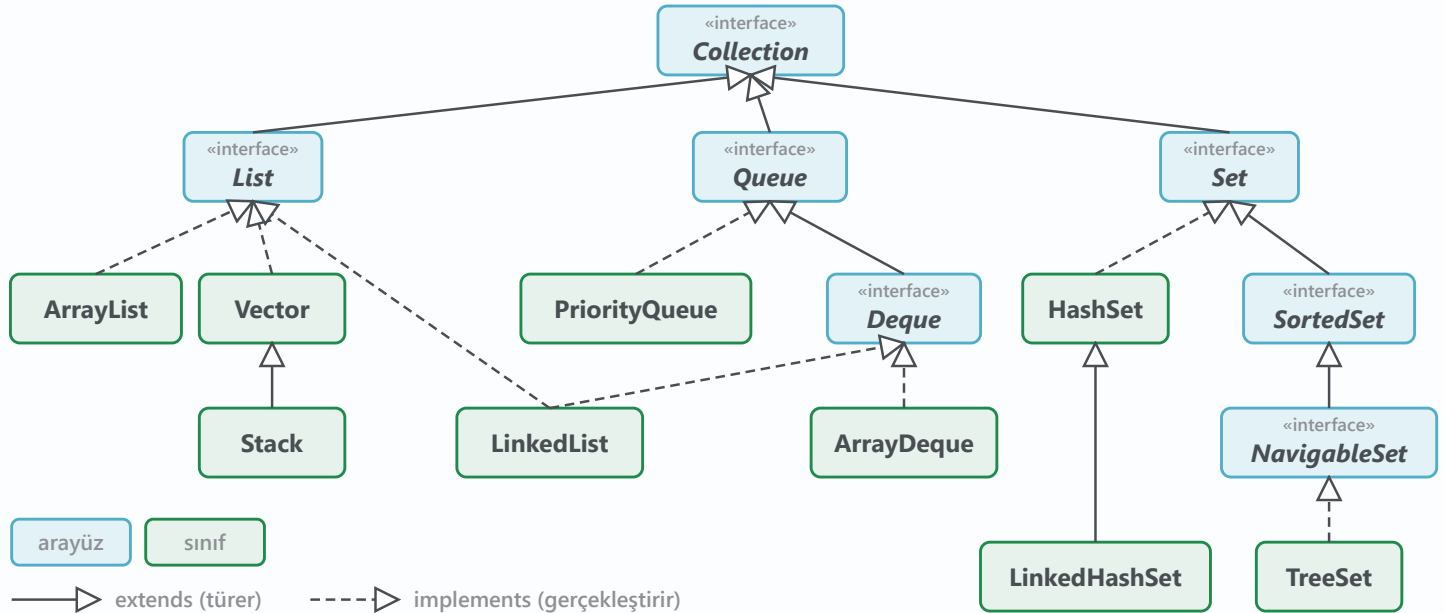
Koleksiyon Arayüzleri



- **Iterable** : `for-each` ile gezilebilen her şey (`iterator()` metodu).
- **Collection** : tüm koleksiyonların ortak metotları (`add` , `remove` , `size` ...).
- **List** , **Queue** , **Set** : **Collection** 'ı genişleten (extends) alt arayüzler.
- **Map** : anahtar-değer yapısı. Çerçevenin parçasıdır ama **Collection** 'dan **türemez**.

Oklar UML gösterimidir: içi boş üçgen, genişletilen (üst) arayüzü gösterir.

Koleksiyon Sınıfları



LinkedList hem **List** hem **Deque** arayüzünü gerçekleştirir. Mavi kutular arayüz (yalnızca davranış), yeşil kutular **new** ile nesnesi oluşturulabilen sınıflardır.

Çerçevenin Üç Unsuru

Arayüz (Interface)

Bir veri yapısının **nasıl davranması gerektiğini** tanımlar (ne yapılır). Örnek: `List` sıralı bir listeyi temsil eder.

Gerçekleştirme (Implementation)

Arayüzü gerçekleştiren gerçek sınıftır (nasıl yapılır). Örnek: `ArrayList`, `List`'i dinamik bir diziyle gerçekleştirir.

Algoritma (Algorithm)

Sıralama, arama gibi işlemleri yapan hazır metotlar. Örnek: `Collections.sort()` bir listeyi sıralar.

```
List<String> liste = new ArrayList<>(); // arayüz ← sınıf
liste.add("Mavi");
liste.add("Kırmızı");
Collections.sort(liste); // algoritma
System.out.println(liste);
```

[Kırmızı, Mavi]

Modüler tasarım: her bileşenin belirli bir rolü ve sorumluluğu vardır. **Esneklik:** farklı uygulamalar ve algoritmalar aynı arayüz üzerinden birlikte kullanılabilir.

Arayüze Göre Programlama

- Arayüz bir **davranış** tanımlar; sınıflar bu davranış kendi metotlarıyla gerçekleştirir.
- Farklı depolama türleri:
 - `ArrayList` dinamik dizi,
 - `LinkedList` çift yönlü bağlı liste, `HashMap` anahtar-değer tablosu.
- Her sınıf belirli bir veri yapısının gereksinimlerini karşılar ve belirli bir görevi yerine getirir.
- Değişkeni ve parametreyi **arayüz tipinde** tanımlayın: uygulamayı tek satırda değiştirebilirsiniz; kodun geri kalanı ve temel davranış aynı kalır.

```
public class ListeYazdir {  
    // Parametre türü arayüz: her List ile çalışır  
    static void yazdir(List<String> liste) {  
        for (String s : liste) {  
            System.out.print(s + " ");  
        }  
        System.out.println("(" + liste.size() + " öge)");  
    }  
  
    public static void main(String[] args) {  
        List<String> a = new ArrayList<>();  
        a.add("Elma");  
        a.add("Armut");  
        List<String> b = new LinkedList<>(a); // aynı öğeler  
        b.add("Kiraz");  
        yazdir(a);  
        yazdir(b);  
    }  
}
```

```
Elma Armut (2 öge)  
Elma Armut Kiraz (3 öge)
```

Jenerik Tipler ve Tip Güvenliği

```
List<String> isimler = new ArrayList<>();
isimler.add("Ali");
// isimler.add(42);           // derleme hatası
String ilk = isimler.get(0); // tür dönüşümü gerekmez

List<Integer> notlar = new ArrayList<>(); // List<int> olmaz
notlar.add(85);                       // kutulama: int → Integer
int n = notlar.get(0);                 // kutu açma: Integer → int
System.out.println(ilk + " " + n);
```

Ali 85

- `<>` içine **öğretilen** yazılır: `List<String>` "String listesi" demektir.
- Yanlış tip derleme zamanında yakalanır; `get` sonucu için tip dönüşümü (cast) gerekmez.
- Sağdaki boş `<>`: elmas (diamond) operatörü, Java 7+; tip soldan çıkarılır.

KOLEKSİYONLAR YALNIZCA NESNE TUTAR

İlkel tip	Sarmalayıcı sınıf
int	Integer
double	Double
char	Character
boolean	Boolean
long	Long

İlkel tipler için sarmalayıcı (wrapper) sınıf kullanılır. Dönüşüm otomatiktir: **otomatik kutulama** (autoboxing) ve **kutu açma** (unboxing).

İlk Örnekler

```
// İsim listesi (ArrayList)
List<String> isimListesi = new ArrayList<>();
isimListesi.add("Ahmet");
isimListesi.add("Ayşe");
System.out.println(isimListesi);

// Telefon rehberi (HashMap)
Map<String, String> telefonRehberi = new HashMap<>();
telefonRehberi.put("Ahmet", "555-1234");
telefonRehberi.put("Ayşe", "555-9876");
System.out.println(telefonRehberi.get("Ahmet"));
```

```
[Ahmet, Ayşe]
555-1234
```

- **İsim listesi:** `ArrayList` öğeleri eklenme sırasıyla tutar; `add` sona ekler.
- **Telefon rehberi:** `HashMap` bir anahtar (isim) bir değere (numara) eşler.
- `put(anahtar, değer)` ile ekleriz, `get(anahtar)` ile değeri buluruz.
- Listeyi `println` ile yazdırmak öğeleri `[ ... ]` içinde gösterir.

Collection Arayüzü Metotları

Metot	Görevi	Döndürür
<code>add(E e)</code>	öğeyi koleksiyona ekler	<code>boolean</code> : eklendiyse <code>true</code>
<code>remove(Object o)</code>	öğeyi koleksiyondan çıkarır	<code>boolean</code>
<code>size()</code>	öge sayısını verir	<code>int</code>
<code>isEmpty()</code>	koleksiyon boş mu?	<code>boolean</code>
<code>contains(Object o)</code>	öge koleksiyonda var mı?	<code>boolean</code>
<code>containsAll(Collection<?> c)</code>	<code>c</code> 'nin tüm öğeleri bu koleksiyonda var mı?	<code>boolean</code>
<code>clear()</code>	tüm öğeleri çıkarır	—
<code>toArray()</code>	koleksiyonu diziye dönüştürür	<code>Object[]</code>
<code>iterator()</code>	gezinmek için bir yineleyici verir	<code>Iterator<E></code>
<code>forEach(Consumer<? super E> a)</code>	her öğeye bir işlem uygular (Java 8+)	—

`E` : öge tipi (jenerik tip parametresi). `?` joker (wildcard): "herhangi bir tip". Bu metotlar tüm `List`, `Set` ve `Queue` 'larda vardır.

Collection Metotları: Örnek

```
Collection<String> renkler = new ArrayList<>();
renkler.add("kırmızı");
renkler.add("yeşil");
renkler.add("mavi");
System.out.println(renkler.size());
System.out.println(renkler.contains("yeşil"));
renkler.remove("yeşil");
System.out.println(renkler);
Collection<String> aranan = Arrays.asList("mavi", "kırmızı");
System.out.println(renkler.containsAll(aranan));
Object[] dizi = renkler.toArray();
System.out.println(dizi.length);
renkler.clear();
System.out.println(renkler.isEmpty());
```

ÇIKTI

```
3
true
[kırmızı, mavi]
true
2
true
```

- Değişken tipi **Collection** : yalnızca ortak metotlar kullanılabilir (indisli **get** yok).
- **remove("yeşil")** öğeyi **değerine** göre siler.
- **toArray()** kopyalanmış yeni bir dizi döndürür.

LİSTELER

List Arayüzü

- Öğeler **eklenme sırasına** göre tutulur: sıralı (ordered) ama sıralanmış (sorted) değil.
- **İndis** tabanlı erişim: ilk öge 0, son öge `size() - 1`.
- Aynı öge birden çok kez bulunabilir.
- **ArrayList** : dinamik dizi, hızlı indisli erişim.
- **LinkedList** : çift yönlü bağlı liste; uçlarda ekleme ve çıkarma hızlı.
- **Vector** : eski, senkronize (thread-safe) sürüm; yerine **ArrayList** tercih edilir.

Metot	Görevi
<code>add(E e)</code>	sona ekler
<code>add(int i, E e)</code>	<code>i</code> konumuna ekler
<code>get(int i)</code>	<code>i</code> 'deki öğeyi verir
<code>set(int i, E e)</code>	<code>i</code> 'deki öğeyi değiştirir
<code>remove(int i)</code>	<code>i</code> 'deki öğeyi çıkarır
<code>remove(Object o)</code>	<code>o</code> 'yu çıkarır
<code>indexOf(Object o)</code>	ilk konumu; yoksa -1
<code>size()</code>	öge sayısı

ArrayList Sınıfı

- **Dinamik dizi:** öğeleri içerideki bir dizide, eklenme sırasıyla saklar.
- **AbstractList** sınıfından türetilmiştir ve **List** arayüzünü gerçekleştirir.
- **Dinamik boyut:** iç dizi dolunca kendiliğinden büyür.
- **get(i)** / **set(i, e)** çok hızlıdır: $O(1)$.
- Sona ekleme ortalamada hızlıdır; **ortaya** ekleme ve ortadan çıkarma öğeleri kaydırıldığı için yavaştır: $O(n)$.

```
ArrayList<String> meyveler = new ArrayList<>();  
meyveler.add("Elma");  
meyveler.add("Armut");  
System.out.println(meyveler.get(1)); // indeksle
```

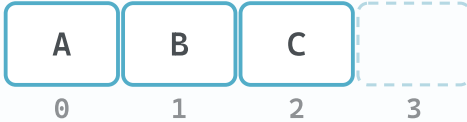
Armut

$O(1)$: süre öğe sayısından bağımsız (sabit). $O(n)$: süre öğe sayısı n ile orantılı.

LİSTELER

ArrayList Nasıl Büyür?

iç dizi (öğeler)



size = 3, kapasite = 4

`new ArrayList<>(4)` ile başlangıç kapasitesi 4 olan bir liste düşünelim: içeride 4 hücrelik bir dizi var, 3'ü dolu.

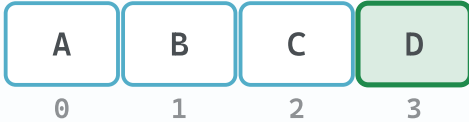
Adım 1 / 4

Varsayılan kapasite 10'dur; her büyümede yaklaşık 1,5 katına çıkar. Kopyalama yalnızca arada bir yapıldığı için sona ekleme **ortalamada** (amortize) $O(1)$ sürer. Kapasite `size()` değildir: `size()` dolu hücre sayısıdır.

LİSTELER

ArrayList Nasıl Büyür?

iç dizi (öğeler)



size = 4, kapasite = 4 (dolu)

`add("D")` : boş hücre var; D doğrudan sona yazılır. Hiçbir öge taşınmaz: $O(1)$.

Adım 2 / 4

Varsayılan kapasite 10'dur; her büyümede yaklaşık 1,5 katına çıkar. Kopyalama yalnızca arada bir yapıldığı için sona ekleme **ortalamada** (amortize) $O(1)$ sürer. Kapasite `size()` değildir: `size()` dolu hücre sayısıdır.

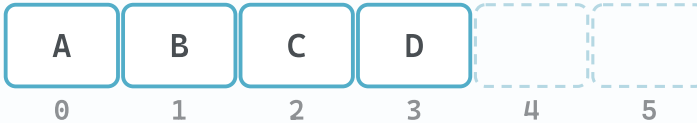
LİSTELER

ArrayList Nasıl Büyür?

iç dizi (öğeler)



4 öğe kopyalanır



yeni dizi: kapasite $4 + 4/2 = 6$

`add("E")` : dizi dolu! Daha büyük yeni bir dizi ayrılır ve eski 4 öge tek tek kopyalanır: $O(n)$.

Adım 3 / 4

Varsayılan kapasite 10'dur; her büyümede yaklaşık 1,5 katına çıkar. Kopyalama yalnızca arada bir yapıldığı için sona ekleme **ortalamada** (amortize) $O(1)$ sürer. Kapasite `size()` değildir: `size()` dolu hücre sayısıdır.

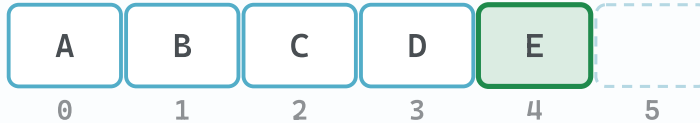
LİSTELER

ArrayList Nasıl Büyür?

iç dizi (öğeler)



eski dizi: artık kullanılmıyor



yeni dizi: kapasite $4 + 4/2 = 6$

size = 5, kapasite = 6

E yeni dizinin 4. hücreğine yazılır. Eski dizi çöp toplayıcıya (garbage collector) bırakılır.

Adım 4 / 4

Varsayılan kapasite 10'dur; her büyümede yaklaşık 1,5 katına çıkar. Kopyalama yalnızca arada bir yapıldığı için sona ekleme **ortalamada** (amortize) $O(1)$ sürer. Kapasite `size()` değildir: `size()` dolu hücre sayısıdır.

ArrayList Metotları: İzleme

```
▶ ArrayList<String> isimler = new ArrayList<>();
2 isimler.add("Ali");
3 isimler.add("Banu");
4 isimler.add("Cem");
5 isimler.add("Deniz");
6 isimler.add("Ece");
7 System.out.println(isimler);
8 isimler.set(3, "Can");      // 3. indeksi
    değiştir
9 System.out.println(isimler);
10 isimler.remove(4);         // 4. indeksi sil
11 System.out.println(isimler);
12 isimler.add(2, "Fatma");    // 2. indekse ekle
13 System.out.println(isimler);
14 int indeks = isimler.indexOf("Banu");
15 System.out.println("indeks: " + indeks);
16 boolean varMi = isimler.contains("Ece");
17 System.out.println("Ece var mı: " + varMi);
18 System.out.println("boyut: " + isimler.size());
```

isimler

size = 0

boş liste

Boş bir `ArrayList` oluştur: `size()` 0.

Adım 1 / 10

DEĞİŞKENLER

—

ÇIKTI

ArrayList Metotları: İzleme

```
1 ArrayList<String> isimler = new ArrayList<>();
2 isimler.add("Ali");
3 isimler.add("Banu");
4 isimler.add("Cem");
5 isimler.add("Deniz");
6 isimler.add("Ece");
7 System.out.println(isimler);
8 isimler.set(3, "Can");      // 3. indeksi
                              değiştir
9 System.out.println(isimler);
10 isimler.remove(4);          // 4. indeksi sil
11 System.out.println(isimler);
12 isimler.add(2, "Fatma");    // 2. indekse ekle
13 System.out.println(isimler);
14 int indeks = isimler.indexOf("Banu");
15 System.out.println("indeks: " + indeks);
16 boolean varMi = isimler.contains("Ece");
17 System.out.println("Ece var mı: " + varMi);
18 System.out.println("boyut: " + isimler.size());
```

isimler

size = 1

Ali

0

add öğeyi sona ekler: Ali 0. indiste.

Adım 2 / 10

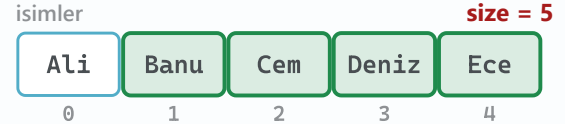
DEĞİŞKENLER

—

ÇIKTI

ArrayList Metotları: İzleme

```
1 ArrayList<String> isimler = new ArrayList<>();
2 isimler.add("Ali");
3 isimler.add("Banu");
4 isimler.add("Cem");
5 isimler.add("Deniz");
6 isimler.add("Ece");
7 System.out.println(isimler);
8 isimler.set(3, "Can"); // 3. indeksi
   değiştir
9 System.out.println(isimler);
10 isimler.remove(4); // 4. indeksi sil
11 System.out.println(isimler);
12 isimler.add(2, "Fatma"); // 2. indekse ekle
13 System.out.println(isimler);
14 int indeks = isimler.indexOf("Banu");
15 System.out.println("indeks: " + indeks);
16 boolean varMi = isimler.contains("Ece");
17 System.out.println("Ece var mı: " + varMi);
18 System.out.println("boyut: " + isimler.size());
```



Dört öğe daha sona eklenir; indisler 0...4.

Adım 3 / 10

DEĞİŞKENLER

—

ÇIKTI

ArrayList Metotları: İzleme

```
1 ArrayList<String> isimler = new ArrayList<>();
2 isimler.add("Ali");
3 isimler.add("Banu");
4 isimler.add("Cem");
5 isimler.add("Deniz");
6 isimler.add("Ece");
7
8 System.out.println(isimler);
9
10 isimler.set(3, "Can"); // 3. indeksi
    değiştir
11 System.out.println(isimler);
12 isimler.remove(4); // 4. indeksi sil
13 System.out.println(isimler);
14 isimler.add(2, "Fatma"); // 2. indekse ekle
15 System.out.println(isimler);
16 int indeks = isimler.indexOf("Banu");
17 System.out.println("indeks: " + indeks);
18 boolean varMi = isimler.contains("Ece");
19 System.out.println("Ece var mı: " + varMi);
20 System.out.println("boyut: " + isimler.size());
```

isimler

size = 5

Ali	Banu	Cem	Deniz	Ece
0	1	2	3	4

Liste yazdırılır: öğeler eklenme sırasıyla.

Adım 4 / 10

DEĞİŞKENLER

—

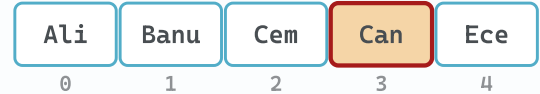
ÇIKTI

[Ali, Banu, Cem, Deniz, Ece]

ArrayList Metotları: İzleme

```
1 ArrayList<String> isimler = new ArrayList<>();
2 isimler.add("Ali");
3 isimler.add("Banu");
4 isimler.add("Cem");
5 isimler.add("Deniz");
6 isimler.add("Ece");
7 System.out.println(isimler);
▶ isimler.set(3, "Can"); // 3. indeksi
  değiştir
▶ System.out.println(isimler);
10 isimler.remove(4); // 4. indeksi sil
11 System.out.println(isimler);
12 isimler.add(2, "Fatma"); // 2. indekse ekle
13 System.out.println(isimler);
14 int indeks = isimler.indexOf("Banu");
15 System.out.println("indeks: " + indeks);
16 boolean varMi = isimler.contains("Ece");
17 System.out.println("Ece var mı: " + varMi);
18 System.out.println("boyut: " + isimler.size());
```

isimler



size = 5

`set(3, "Can")` : 3. indisteki Deniz'in yerine Can yazılır. Boyut değişmez.

Adım 5 / 10

DEĞİŞKENLER

—

ÇIKTI

```
[Ali, Banu, Cem, Deniz, Ece]
[Ali, Banu, Cem, Can, Ece]
```

ArrayList Metotları: İzleme

```
1 ArrayList<String> isimler = new ArrayList<>();
2 isimler.add("Ali");
3 isimler.add("Banu");
4 isimler.add("Cem");
5 isimler.add("Deniz");
6 isimler.add("Ece");
7 System.out.println(isimler);
8 isimler.set(3, "Can");      // 3. indeksi
   değiştir
9 System.out.println(isimler);
10 ▶ isimler.remove(4);        // 4. indeksi sil
11 ▶ System.out.println(isimler);
12 isimler.add(2, "Fatma");    // 2. indekse ekle
13 System.out.println(isimler);
14 int indeks = isimler.indexOf("Banu");
15 System.out.println("indeks: " + indeks);
16 boolean varMi = isimler.contains("Ece");
17 System.out.println("Ece var mı: " + varMi);
18 System.out.println("boyut: " + isimler.size());
```

isimler

size = 4

Ali	Banu	Cem	Can
0	1	2	3

remove(4) : 4. indisteki Ece silinir. Son öğe olduğu için kaydırma gerekmez.

Adım 6 / 10

DEĞİŞKENLER

—

ÇIKTI

```
[Ali, Banu, Cem, Deniz, Ece]
[Ali, Banu, Cem, Can, Ece]
[Ali, Banu, Cem, Can]
```

ArrayList Metotları: İzleme

```
1 ArrayList<String> isimler = new ArrayList<>();
2 isimler.add("Ali");
3 isimler.add("Banu");
4 isimler.add("Cem");
5 isimler.add("Deniz");
6 isimler.add("Ece");
7 System.out.println(isimler);
8 isimler.set(3, "Can");          // 3. indeksi
    değiştir
9 System.out.println(isimler);
10 isimler.remove(4);             // 4. indeksi sil
11 System.out.println(isimler);
    ► isimler.add(2, "Fatma");    // 2. indekse ekle
    ► System.out.println(isimler);
14 int indeks = isimler.indexOf("Banu");
15 System.out.println("indeks: " + indeks);
16 boolean varMi = isimler.contains("Ece");
17 System.out.println("Ece var mı: " + varMi);
18 System.out.println("boyut: " + isimler.size());
```

isimler



size = 5

`add(2, "Fatma")` : Cem ve Can birer sağa kayar, Fatma 2. indise girer.

Adım 7 / 10

DEĞİŞKENLER

—

ÇIKTI

```
[Ali, Banu, Cem, Deniz, Ece]
[Ali, Banu, Cem, Can, Ece]
[Ali, Banu, Cem, Can]
[Ali, Banu, Fatma, Cem, Can]
```

ArrayList Metotları: İzleme

```
1 ArrayList<String> isimler = new ArrayList<>();
2 isimler.add("Ali");
3 isimler.add("Banu");
4 isimler.add("Cem");
5 isimler.add("Deniz");
6 isimler.add("Ece");
7 System.out.println(isimler);
8 isimler.set(3, "Can");      // 3. indeksi
    değiştir
9 System.out.println(isimler);
10 isimler.remove(4);          // 4. indeksi sil
11 System.out.println(isimler);
12 isimler.add(2, "Fatma");    // 2. indekse ekle
13 System.out.println(isimler);
14 ▶ int indeks = isimler.indexOf("Banu");
15 ▶ System.out.println("indeks: " + indeks);
16 boolean varMi = isimler.contains("Ece");
17 System.out.println("Ece var mı: " + varMi);
18 System.out.println("boyut: " + isimler.size());
```

isimler



`indexOf("Banu")` baştan arar ve ilk konumu döndürür: 1.

Adım 8 / 10

DEĞİŞKENLER

`indeks = 1`

ÇIKTI

```
[Ali, Banu, Cem, Deniz, Ece]
[Ali, Banu, Cem, Can, Ece]
[Ali, Banu, Cem, Can]
[Ali, Banu, Fatma, Cem, Can]
indeks: 1
```

ArrayList Metotları: İzleme

```
1 ArrayList<String> isimler = new ArrayList<>();
2 isimler.add("Ali");
3 isimler.add("Banu");
4 isimler.add("Cem");
5 isimler.add("Deniz");
6 isimler.add("Ece");
7 System.out.println(isimler);
8 isimler.set(3, "Can");      // 3. indeksi
                             // değiştir
9 System.out.println(isimler);
10 isimler.remove(4);         // 4. indeksi sil
11 System.out.println(isimler);
12 isimler.add(2, "Fatma");   // 2. indekse ekle
13 System.out.println(isimler);
14 int indeks = isimler.indexOf("Banu");
15 System.out.println("indeks: " + indeks);
16 ▶ boolean varMi = isimler.contains("Ece");
17 ▶ System.out.println("Ece var mı: " + varMi);
18 System.out.println("boyut: " + isimler.size());
```

isimler

size = 5

Ali	Banu	Fatma	Cem	Can
0	1	2	3	4

`contains("Ece")` : Ece daha önce silindi,
sonuç `false`.

Adım 9 / 10

DEĞİŞKENLER

indeks = 1

varMi = false

ÇIKTI

```
[Ali, Banu, Cem, Deniz, Ece]
[Ali, Banu, Cem, Can, Ece]
[Ali, Banu, Cem, Can]
[Ali, Banu, Fatma, Cem, Can]
indeks: 1
Ece var mı: false
```

ArrayList Metotları: İzleme

```
1 ArrayList<String> isimler = new ArrayList<>();
2 isimler.add("Ali");
3 isimler.add("Banu");
4 isimler.add("Cem");
5 isimler.add("Deniz");
6 isimler.add("Ece");
7 System.out.println(isimler);
8 isimler.set(3, "Can");      // 3. indeksi
                             // değiştir
9 System.out.println(isimler);
10 isimler.remove(4);         // 4. indeksi sil
11 System.out.println(isimler);
12 isimler.add(2, "Fatma");    // 2. indekse ekle
13 System.out.println(isimler);
14 int indeks = isimler.indexOf("Banu");
15 System.out.println("indeks: " + indeks);
16 boolean varMi = isimler.contains("Ece");
17 System.out.println("Ece var mı: " + varMi);
18 System.out.println("boyut: " + isimler.size());
```

isimler



size = 5

Son boyut 5: 5 ekleme + 1 araya ekleme – 1 silme.

Adım 10 / 10

DEĞİŞKENLER

indeks = 1

varMi = false

ÇIKTI

```
[Ali, Banu, Cem, Deniz, Ece]
[Ali, Banu, Cem, Can, Ece]
[Ali, Banu, Cem, Can]
[Ali, Banu, Fatma, Cem, Can]
indeks: 1
Ece var mı: false
boyut: 5
```

LİSTELER

Ortaya Ekleme ve Silme

```
List<Integer> liste2 = new ArrayList<>(Arrays.asList(10, 20, 30, 40));  
liste2.add(1, 15);      // araya ekle: 20, 30, 40 sağa kayar  
System.out.println(liste2);  
liste2.remove(0);       // baştan sil: hepsi sola kayar  
System.out.println(liste2);
```



Başlangıç: `[10, 20, 30, 40]` . Şimdi `add(1, 15)` çalışacak.

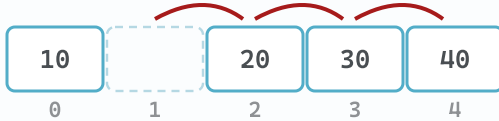
Adım 1 / 5

```
[10, 15, 20, 30, 40]  
[15, 20, 30, 40]
```

LİSTELER

Ortaya Ekleme ve Silme

```
List<Integer> liste2 = new ArrayList<>(Arrays.asList(10, 20, 30, 40));  
liste2.add(1, 15);      // araya ekle: 20, 30, 40 sağa kayar  
System.out.println(liste2);  
liste2.remove(0);       // baştan sil: hepsi sola kayar  
System.out.println(liste2);
```



3 öğe sağa

Önce 1. indisten sonraki öğeler (sondan başlayarak) birer sağa kayar; 1. hücre boşalır.

Adım 2 / 5

[10, 15, 20, 30, 40]

[15, 20, 30, 40]

LİSTELER

Ortaya Ekleme ve Silme

```
List<Integer> liste2 = new ArrayList<>(Arrays.asList(10, 20, 30, 40));  
liste2.add(1, 15);      // araya ekle: 20, 30, 40 sağa kayar  
System.out.println(liste2);  
liste2.remove(0);      // baştan sil: hepsi sola kayar  
System.out.println(liste2);
```



[10, 15, 20, 30, 40]

15 boşalan hücreye yazılır. n öğeli listede en kötü durumda n kaydırma: $O(n)$.

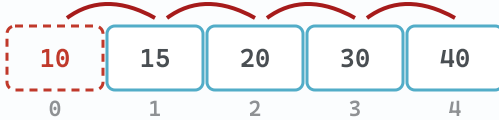
Adım 3 / 5

```
[10, 15, 20, 30, 40]  
[15, 20, 30, 40]
```

LİSTELER

Ortaya Ekleme ve Silme

```
List<Integer> liste2 = new ArrayList<>(Arrays.asList(10, 20, 30, 40));
liste2.add(1, 15);      // araya ekle: 20, 30, 40 sağa kayar
System.out.println(liste2);
liste2.remove(0);       // baştan sil: hepsi sola kayar
System.out.println(liste2);
```



4 öge sola

`remove(0)` : 10 silinir, arkasındaki 4 öge birer sola kayar.

Adım 4 / 5

[10, 15, 20, 30, 40]

[15, 20, 30, 40]

LİSTELER

Ortaya Ekleme ve Silme

```
List<Integer> liste2 = new ArrayList<>(Arrays.asList(10, 20, 30, 40));  
liste2.add(1, 15);      // araya ekle: 20, 30, 40 sağa kayar  
System.out.println(liste2);  
liste2.remove(0);       // baştan sil: hepsi sola kayar  
System.out.println(liste2);
```



[15, 20, 30, 40]

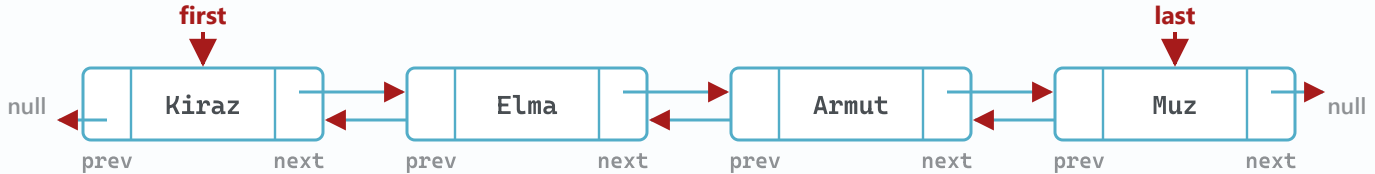
Sonuç. Sondan silmede kaydırma gerekmez: $O(1)$. Baştan/ortadan silmede $O(n)$.

Adım 5 / 5

```
[10, 15, 20, 30, 40]  
[15, 20, 30, 40]
```

LinkedList Sınıfı

- **Çift yönlü bağlı liste:** öğeler **düğüm**lerde (node) saklanır; her düğüm değeri ve önceki (prev) ile sonraki (next) düğümün referansını tutar.
- `AbstractSequentialList` sınıfından türetilmiştir; `List` ve `Deque` arayüzlerini gerçekleştirir.
- **Esnek boyut:** kapasite yoktur; her ekleme yeni bir düğüm oluşturur, kaydırma yapılmaz.
- Uçlarda ekleme ve çıkarma $O(1)$; ancak `get(i)` düğümleri tek tek yürür: $O(n)$.



Benzetme: el ele tutuşmuş bir sıra insan. Araya biri girecekse yalnızca iki kişi el değiştirir; ama 3. kişiyi bulmak için baştan saymak gerekir.

LİSTELER

LinkedList Örneği

```
LinkedList<String> meyveler = new LinkedList<>>();
meyveler.add("Elma");
meyveler.add("Armut");
meyveler.addFirst("Kiraz"); // başa ekle
meyveler.addLast("Muz");    // sona ekle
System.out.println(meyveler);
System.out.println(meyveler.getFirst());
System.out.println(meyveler.getLast());
meyveler.removeFirst();
System.out.println(meyveler);
```

```
[Kiraz, Elma, Armut, Muz]
Kiraz
Muz
[Elma, Armut, Muz]
```

Metot	Görevi
<code>addFirst(e)</code>	baş a ekler
<code>addLast(e)</code>	sona ekler
<code>getFirst()</code>	ilk öğeyi verir
<code>getLast()</code>	son öğeyi verir
<code>removeFirst()</code>	ilk öğeyi çıkarıp verir
<code>removeLast()</code>	son öğeyi çıkarıp verir

Bu metotlar `Deque` arayüzünden gelir; `add` , `get` , `remove(i)` ise `List` 'ten.

LinkedList: Düğümler ve Bağlar

```
► LinkedList<String> meyveler = new LinkedList<>();  
2 meyveler.add("Elma");  
3 meyveler.add("Armut");  
4 meyveler.addFirst("Kiraz");  
5 meyveler.addLast("Muz");  
6 System.out.println(meyveler);  
7 System.out.println(meyveler.getFirst());  
8 System.out.println(meyveler.getLast());  
9 meyveler.removeFirst();  
10 System.out.println(meyveler);
```

`first = null`

`last = null`

Boş liste: `first` ve `last` referansları `null`. Kapasite diye bir şey yok.

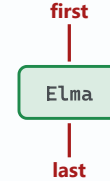
Adım 1 / 10

ÇIKTI



LinkedList: Düğümler ve Bağlar

```
1 LinkedList<String> meyveler = new LinkedList<>();  
2 ► meyveler.add("Elma");  
3 meyveler.add("Armut");  
4 meyveler.addFirst("Kiraz");  
5 meyveler.addLast("Muz");  
6 System.out.println(meyveler);  
7 System.out.println(meyveler.getFirst());  
8 System.out.println(meyveler.getLast());  
9 meyveler.removeFirst();  
10 System.out.println(meyveler);
```



Elma için bir düğüm oluşur; tek düğüm hem ilk hem son.

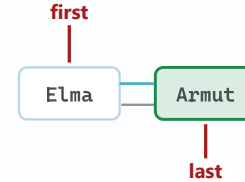
Adım 2 / 10

ÇIKTI



LinkedList: Düğümler ve Bağlar

```
1 LinkedList<String> meyveler = new LinkedList<>();
2 meyveler.add("Elma");
3 ► meyveler.add("Armut");
4 meyveler.addFirst("Kiraz");
5 meyveler.addLast("Muz");
6 System.out.println(meyveler);
7 System.out.println(meyveler.getFirst());
8 System.out.println(meyveler.getLast());
9 meyveler.removeFirst();
10 System.out.println(meyveler);
```



Armut sona bağlanır: Elma.next → Armut,
Armut.prev → Elma; **last** güncellenir.

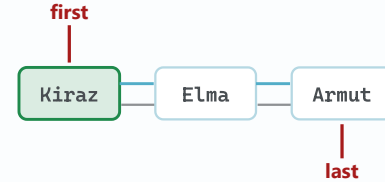
Adım 3 / 10

ÇIKTI



LinkedList: Düğümler ve Bağlar

```
1 LinkedList<String> meyveler = new LinkedList<>();
2 meyveler.add("Elma");
3 meyveler.add("Armut");
4 ► meyveler.addFirst("Kiraz");
5 meyveler.addLast("Muz");
6 System.out.println(meyveler);
7 System.out.println(meyveler.getFirst());
8 System.out.println(meyveler.getLast());
9 meyveler.removeFirst();
10 System.out.println(meyveler);
```



addFirst : Kiraz başa; yalnızca birkaç referans değişir, öge kaydırılmaz: $O(1)$.

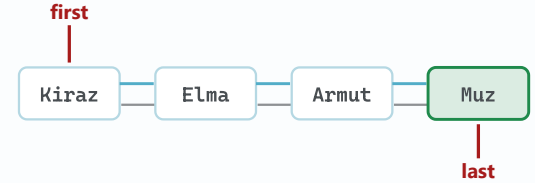
Adım 4 / 10

ÇIKTI



LinkedList: Düğümler ve Bağlar

```
1 LinkedList<String> meyveler = new LinkedList<>();
2 meyveler.add("Elma");
3 meyveler.add("Armut");
4 meyveler.addFirst("Kiraz");
5 ► meyveler.addLast("Muz");
6 System.out.println(meyveler);
7 System.out.println(meyveler.getFirst());
8 System.out.println(meyveler.getLast());
9 meyveler.removeFirst();
10 System.out.println(meyveler);
```



addLast : Muz sona, yine $O(1)$.

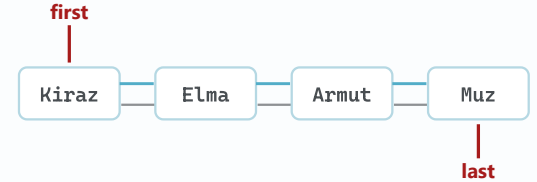
Adım 5 / 10

ÇIKTI



LinkedList: Düğümler ve Bağlar

```
1 LinkedList<String> meyveler = new LinkedList<>();
2 meyveler.add("Elma");
3 meyveler.add("Armut");
4 meyveler.addFirst("Kiraz");
5 meyveler.addLast("Muz");
6 ► System.out.println(meyveler);
7 System.out.println(meyveler.getFirst());
8 System.out.println(meyveler.getLast());
9 meyveler.removeFirst();
10 System.out.println(meyveler);
```



println listeyi baştan sona **next** bağlantıyla gezerek yazar.

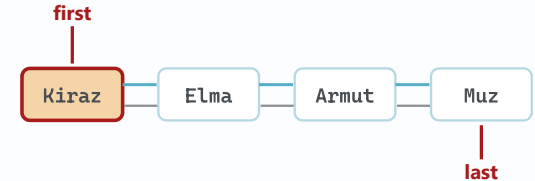
Adım 6 / 10

ÇIKTI

```
[Kiraz, Elma, Armut, Muz]
```

LinkedList: Düğümler ve Bağlar

```
1 LinkedList<String> meyveler = new LinkedList<>();
2 meyveler.add("Elma");
3 meyveler.add("Armut");
4 meyveler.addFirst("Kiraz");
5 meyveler.addLast("Muz");
6 System.out.println(meyveler);
7 ► System.out.println(meyveler.getFirst());
8 System.out.println(meyveler.getLast());
9 meyveler.removeFirst();
10 System.out.println(meyveler);
```



`getFirst()` : `first` 'ün gösterdiği düğüm, Kiraz.

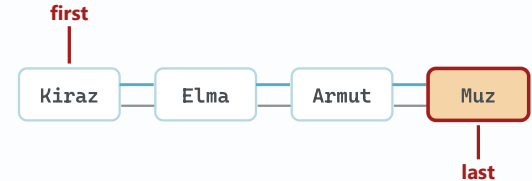
Adım 7 / 10

ÇIKTI

```
[Kiraz, Elma, Armut, Muz]
Kiraz
```

LinkedList: Düğümler ve Bağlar

```
1 LinkedList<String> meyveler = new LinkedList<>();
2 meyveler.add("Elma");
3 meyveler.add("Armut");
4 meyveler.addFirst("Kiraz");
5 meyveler.addLast("Muz");
6 System.out.println(meyveler);
7 System.out.println(meyveler.getFirst());
8 ► System.out.println(meyveler.getLast());
9 meyveler.removeFirst();
10 System.out.println(meyveler);
```



`getLast()` : `last` 'ün gösterdiği düğüm, Muz. Ortadaki bir öge için (`get(i)`) ise düğüm düğüm yürünür.

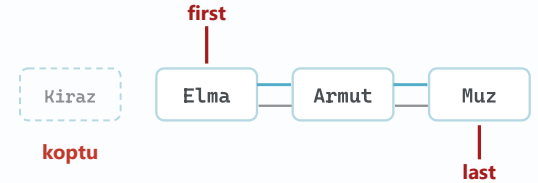
Adım 8 / 10

ÇIKTI

```
[Kiraz, Elma, Armut, Muz]
Kiraz
Muz
```

LinkedList: Düğümler ve Bağlar

```
1 LinkedList<String> meyveler = new LinkedList<>();
2 meyveler.add("Elma");
3 meyveler.add("Armut");
4 meyveler.addFirst("Kiraz");
5 meyveler.addLast("Muz");
6 System.out.println(meyveler);
7 System.out.println(meyveler.getFirst());
8 System.out.println(meyveler.getLast());
9 ► meyveler.removeFirst();
10 System.out.println(meyveler);
```



`removeFirst()` : `first` Elma'ya geçer, Elma.prev null olur; Kiraz düğümü listeden kopar.

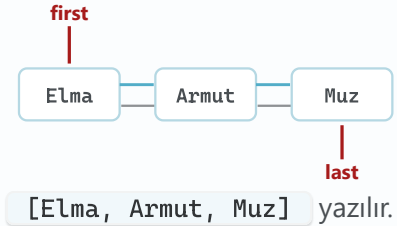
Adım 9 / 10

ÇIKTI

```
[Kiraz, Elma, Armut, Muz]
Kiraz
Muz
```

LinkedList: Düğümler ve Bağlar

```
1 LinkedList<String> meyveler = new LinkedList<>();
2 meyveler.add("Elma");
3 meyveler.add("Armut");
4 meyveler.addFirst("Kiraz");
5 meyveler.addLast("Muz");
6 System.out.println(meyveler);
7 System.out.println(meyveler.getFirst());
8 System.out.println(meyveler.getLast());
9 meyveler.removeFirst();
▶ System.out.println(meyveler);
```



Adım 10 / 10

ÇIKTI

```
[Kiraz, Elma, Armut, Muz]
Kiraz
Muz
[Elma, Armut, Muz]
```

LİSTELER

ArrayList mi, LinkedList mi?

İşlem	ArrayList	LinkedList
<code>get(i)</code> , <code>set(i, e)</code>	O(1)	O(n): düğüm düğüm yürür
Sona ekleme <code>add(e)</code>	O(1) ortalama (arada bir kopyalama)	O(1)
Baş ekleme / baştan silme	O(n): tüm öğeler kayar	O(1)
Ortaya ekleme / ortadan silme	O(n): kaydırma	O(n): konumu bulma + O(1) bağlama
Bellek kullanımı	az: yalnızca dizi	fazla: her düğümde 2 ek referans

Çoğu durumda **ArrayList** seçin. Öğ ekleme ve çıkarma işlemleri sık ve özellikle listenin **uçlarında** ise **LinkedList** (ya da kuyruk/yığın için **ArrayDeque**) tercih edilebilir.

Vector Sınıfı

- **List** arayüzünü gerçekleştiren, genişletilebilir bir dinamik dizidir.
- Belirtilen (varsayılan 10) kapasiteyle başlar; doldukça kapasitesi **iki katına** çıkar.
- Metotları senkronizedir (synchronized): çoklu iş parçacıklı ortamda güvenli kullanılabilir; ama bu her çağrıda ek maliyettir.
- Java 1.0'dan kalmadır; modern kodda daha hafif ve hızlı **ArrayList** tercih edilir.

```
Vector<Integer> v = new Vector<>(); // kapasite: 10
for (int i = 1; i <= 11; i++) {
    v.addElement(i * 10);
}
System.out.println(v.size() + " / " + v.capacity());
System.out.println(v.elementAt(0));
v.removeElement(110); // değeri 110 olan öge
System.out.println(v.size() + " / " + v.capacity());
```

```
11 / 20
10
10 / 20
```

11. öge eklenince kapasite 10 → 20 olur. Silme kapasiteyi küçültmez.

addElement(E obj)	elementAt(int i)	removeElement(Object obj)	size()	capacity()
öge ekler	i 'deki ögeyi verir	ögeyi çıkarır	öge sayısı	mevcut kapasite

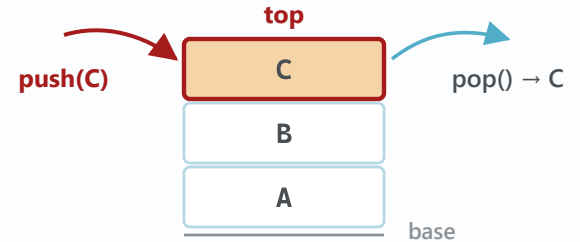
LİSTELER

Stack Sınıfı

- **LIFO** (Last In, First Out): son giren ilk çıkar.
- En son eklenen en üstte (top), en eski eklenen en altta (base) durur.
- **Vector** sınıfından türetilmiştir (genişletilmiş bir sürümüdür).
- Kullanım: geri alma (undo), metin ve ifade analizi (parantez denetimi), derinlik öncelikli arama, metot çağrı yığını.
- Modern kodda yerine **Deque** arayüzünü gerçekleştiren **ArrayDeque** (veya **LinkedList**) önerilir.

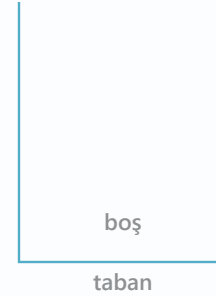
Benzetme: üst üste konmuş tabaklar; tabak hep en üstten alınır ve en üste konur.

Metot	Görevi
<code>push(E item)</code>	en üste ekler
<code>pop()</code>	en üsttekini çıkarır ve döndürür
<code>peek()</code>	en üsttekini döndürür, çıkarmaz
<code>empty()</code>	yığın boş mu?
<code>search(Object o)</code>	üstten 1'den başlayan konum; yoksa -1



Stack: İzleme

```
► Stack<String> yigin = new Stack<>();  
2 yigin.push("A");  
3 yigin.push("B");  
4 yigin.push("C");  
5 System.out.println(yigin);  
6 System.out.println(yigin.peek());  
7 System.out.println(yigin.pop());  
8 System.out.println(yigin.search("A"));  
9 System.out.println(yigin.empty());
```



Boş bir yığın oluşur.

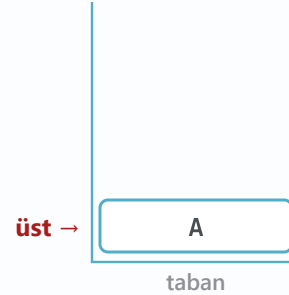
Adım 1 / 9

ÇIKTI



Stack: İzleme

```
1 Stack<String> yigin = new Stack<>();  
▶ yigin.push("A");  
3 yigin.push("B");  
4 yigin.push("C");  
5 System.out.println(yigin);  
6 System.out.println(yigin.peek());  
7 System.out.println(yigin.pop());  
8 System.out.println(yigin.search("A"));  
9 System.out.println(yigin.empty());
```



push("A") : A en üste konur.

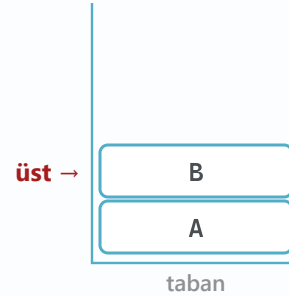
Adım 2 / 9

ÇIKTI



Stack: İzleme

```
1 Stack<String> yigin = new Stack<>();  
2 yigin.push("A");  
3 ► yigin.push("B");  
4 yigin.push("C");  
5 System.out.println(yigin);  
6 System.out.println(yigin.peek());  
7 System.out.println(yigin.pop());  
8 System.out.println(yigin.search("A"));  
9 System.out.println(yigin.empty());
```



push("B") : B, A'nın üstüne.

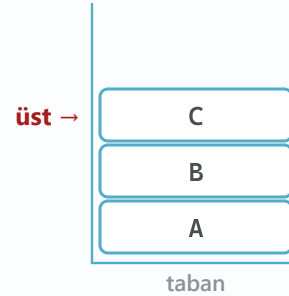
Adım 3 / 9

ÇIKTI



Stack: İzleme

```
1 Stack<String> yigin = new Stack<>();  
2 yigin.push("A");  
3 yigin.push("B");  
▶ yigin.push("C");  
5 System.out.println(yigin);  
6 System.out.println(yigin.peek());  
7 System.out.println(yigin.pop());  
8 System.out.println(yigin.search("A"));  
9 System.out.println(yigin.empty());
```



push("C") : en üstte artık C var.

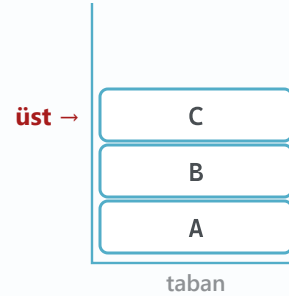
Adım 4 / 9

ÇIKTI



Stack: İzleme

```
1 Stack<String> yigin = new Stack<>();  
2 yigin.push("A");  
3 yigin.push("B");  
4 yigin.push("C");  
▶ System.out.println(yigin);  
6 System.out.println(yigin.peek());  
7 System.out.println(yigin.pop());  
8 System.out.println(yigin.search("A"));  
9 System.out.println(yigin.empty());
```



`println` öğeleri **alttan üste** yazar: `[A, B, C]`.

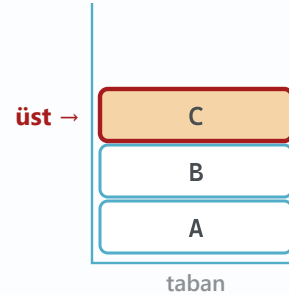
Adım 5 / 9

ÇIKTI

[A, B, C]

Stack: İzleme

```
1 Stack<String> yigin = new Stack<>();  
2 yigin.push("A");  
3 yigin.push("B");  
4 yigin.push("C");  
5 System.out.println(yigin);  
▶ System.out.println(yigin.peek());  
7 System.out.println(yigin.pop());  
8 System.out.println(yigin.search("A"));  
9 System.out.println(yigin.empty());
```



`peek()` en üsttekini (C) döndürür ama çıkarmaz.

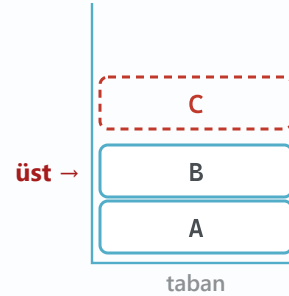
Adım 6 / 9

ÇIKTI

```
[A, B, C]  
C
```

Stack: İzleme

```
1 Stack<String> yigin = new Stack<>();  
2 yigin.push("A");  
3 yigin.push("B");  
4 yigin.push("C");  
5 System.out.println(yigin);  
6 System.out.println(yigin.peek());  
7 System.out.println(yigin.pop());  
8 System.out.println(yigin.search("A"));  
9 System.out.println(yigin.empty());
```



pop() C'yi çıkarır ve döndürür: son giren ilk çıktı.

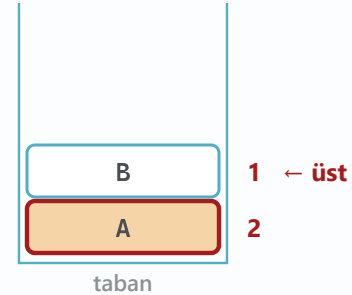
Adım 7 / 9

ÇIKTI

```
[A, B, C]  
C  
C
```

Stack: İzleme

```
1 Stack<String> yigin = new Stack<>();  
2 yigin.push("A");  
3 yigin.push("B");  
4 yigin.push("C");  
5 System.out.println(yigin);  
6 System.out.println(yigin.peek());  
7 System.out.println(yigin.pop());  
▶ System.out.println(yigin.search("A"));  
9 System.out.println(yigin.empty());
```



search("A") konumu **üstten 1'den** sayar: B 1, A 2.

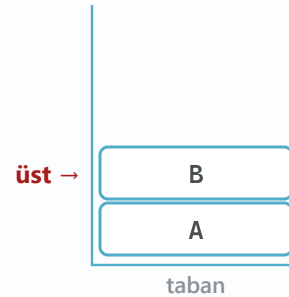
Adım 8 / 9

ÇIKTI

```
[A, B, C]  
C  
C  
2
```

Stack: İzleme

```
1 Stack<String> yigin = new Stack<>();  
2 yigin.push("A");  
3 yigin.push("B");  
4 yigin.push("C");  
5 System.out.println(yigin);  
6 System.out.println(yigin.peek());  
7 System.out.println(yigin.pop());  
8 System.out.println(yigin.search("A"));  
▶ System.out.println(yigin.empty());
```



`empty()` : yığında 2 öge var, sonuç `false` .

Adım 9 / 9

ÇIKTI

```
[A, B, C]  
C  
C  
2  
false
```

LİSTELER

Listede Gezinme

```
List<String> dersler = new ArrayList<>(  
    Arrays.asList("Fizik", "Java", "Devre"));  
// 1) İndeksli for  
for (int i = 0; i < dersler.size(); i++) {  
    System.out.print(dersler.get(i) + " ");  
}  
System.out.println();  
// 2) Gelişmiş for (for-each)  
for (String d : dersler) {  
    System.out.print(d + " ");  
}  
System.out.println();  
// 3) forEach + lambda (Java 8+)  
dersler.forEach(d -> System.out.print(d + " "));  
System.out.println();
```

ÇIKTI

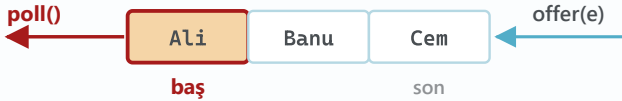
```
Fizik Java Devre  
Fizik Java Devre  
Fizik Java Devre
```

- **İndisli for:** indise ihtiyaç olduğunda (yalnızca listelerde).
- **for-each:** en okunur yol; tüm koleksiyonlarda çalışır.
- **forEach + lambda:** Java 8+ kısa yazım.
- Gezinirken öge silmek gerekiyorsa **Iterator** kullanılır (ileride).

KUYRUKLAR

Queue Arayüzü

- **FIFO** (First-In-First-Out): ilk giren ilk çıkar.
- Öğeler kuyruğa eklendikleri sırayla çıkarılır; yeni öğe **sona** girer, işlenecek öğe **baştan** alınır.
- Öğeler genellikle `offer`, `poll` ve `peek` metotlarıyla işlenir.
- Uygulamalar: `LinkedList` (çift yönlü bağlı liste), `PriorityQueue` (önceliğe göre; FIFO değil), `ArrayDeque` (dinamik dizi).
- Kullanım: yazıcı kuyruğu, görev sırası, genişlik öncelikli arama.



Benzetme: market kasası. Sıraya en sondan girilir, ödemeyi en öndeki yapar.

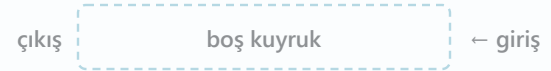
İşlem	Hata fırlatır	Özel değer döner
Ekle (sona)	<code>add(e)</code>	<code>offer(e)</code> → <code>false</code>
Çıkar (baştan)	<code>remove()</code>	<code>poll()</code> → <code>null</code>
Bak (baştakine)	<code>element()</code>	<code>peek()</code> → <code>null</code>

Kuyruk boşken `remove()` ve `element()`

`NoSuchElementException` fırlatır; `poll()` ve `peek()` ise `null` döndürür. `size()` : öğe sayısı.

Queue: İzleme

```
► Queue<String> kuyruk = new LinkedList<>();  
2 kuyruk.offer("Ali");  
3 kuyruk.offer("Banu");  
4 kuyruk.offer("Cem");  
5 System.out.println(kuyruk);  
6 System.out.println(kuyruk.peek());  
7 System.out.println(kuyruk.poll());  
8 System.out.println(kuyruk.poll());  
9 System.out.println(kuyruk.size());  
10 System.out.println(kuyruk.poll());  
11 System.out.println(kuyruk.poll());
```



Kuyruk `Queue` tipinde, nesne `LinkedList`.

Adım 1 / 11

ÇIKTI



Queue: İzleme

```
1 Queue<String> kuyruk = new LinkedList<>();
2 kuyruk.offer("Ali");
3 kuyruk.offer("Banu");
4 kuyruk.offer("Cem");
5 System.out.println(kuyruk);
6 System.out.println(kuyruk.peek());
7 System.out.println(kuyruk.poll());
8 System.out.println(kuyruk.poll());
9 System.out.println(kuyruk.size());
10 System.out.println(kuyruk.poll());
11 System.out.println(kuyruk.poll());
```

çıkış

Ali

← giriş

baş = son

`offer("Ali")` : Ali kuyruğa girer; hem baş hem son o.

Adım 2 / 11

ÇIKTI



Queue: İzleme

```
1 Queue<String> kuyruk = new LinkedList<>();
2 kuyruk.offer("Ali");
3 ► kuyruk.offer("Banu");
4 kuyruk.offer("Cem");
5 System.out.println(kuyruk);
6 System.out.println(kuyruk.peek());
7 System.out.println(kuyruk.poll());
8 System.out.println(kuyruk.poll());
9 System.out.println(kuyruk.size());
10 System.out.println(kuyruk.poll());
11 System.out.println(kuyruk.poll());
```



Banu sona girer.

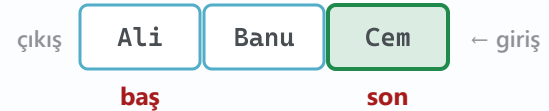
Adım 3 / 11

ÇIKTI



Queue: İzleme

```
1 Queue<String> kuyruk = new LinkedList<>();
2 kuyruk.offer("Ali");
3 kuyruk.offer("Banu");
4 ► kuyruk.offer("Cem");
5 System.out.println(kuyruk);
6 System.out.println(kuyruk.peek());
7 System.out.println(kuyruk.poll());
8 System.out.println(kuyruk.poll());
9 System.out.println(kuyruk.size());
10 System.out.println(kuyruk.poll());
11 System.out.println(kuyruk.poll());
```



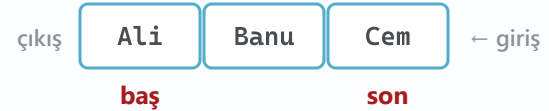
Cem sona girer.

Adım 4 / 11

ÇIKTI

Queue: İzleme

```
1 Queue<String> kuyruk = new LinkedList<>();
2 kuyruk.offer("Ali");
3 kuyruk.offer("Banu");
4 kuyruk.offer("Cem");
5 ▶ System.out.println(kuyruk);
6 System.out.println(kuyruk.peek());
7 System.out.println(kuyruk.poll());
8 System.out.println(kuyruk.poll());
9 System.out.println(kuyruk.size());
10 System.out.println(kuyruk.poll());
11 System.out.println(kuyruk.poll());
```



Kuyruk baştan sona yazdırılır.

Adım 5 / 11

ÇIKTI

[Ali, Banu, Cem]

Queue: İzleme

```
1 Queue<String> kuyruk = new LinkedList<>();
2 kuyruk.offer("Ali");
3 kuyruk.offer("Banu");
4 kuyruk.offer("Cem");
5 System.out.println(kuyruk);
6 ► System.out.println(kuyruk.peek());
7 System.out.println(kuyruk.poll());
8 System.out.println(kuyruk.poll());
9 System.out.println(kuyruk.size());
10 System.out.println(kuyruk.poll());
11 System.out.println(kuyruk.poll());
```



`peek()` baştakini döndürür, çıkarmaz.

Adım 6 / 11

ÇIKTI

```
[Ali, Banu, Cem]
Ali
```

Queue: İzleme

```
1 Queue<String> kuyruk = new LinkedList<>();
2 kuyruk.offer("Ali");
3 kuyruk.offer("Banu");
4 kuyruk.offer("Cem");
5 System.out.println(kuyruk);
6 System.out.println(kuyruk.peek());
7 ► System.out.println(kuyruk.poll());
8 System.out.println(kuyruk.poll());
9 System.out.println(kuyruk.size());
10 System.out.println(kuyruk.poll());
11 System.out.println(kuyruk.poll());
```

Ali

çıkış

Banu

Cem

← giriş

baş

son

poll() baştakini çıkarır ve döndürür: ilk giren Ali ilk çıktı.

Adım 7 / 11

ÇIKTI

[Ali, Banu, Cem]

Ali

Ali

Queue: İzleme

```
1 Queue<String> kuyruk = new LinkedList<>();
2 kuyruk.offer("Ali");
3 kuyruk.offer("Banu");
4 kuyruk.offer("Cem");
5 System.out.println(kuyruk);
6 System.out.println(kuyruk.peek());
7 System.out.println(kuyruk.poll());
8 ► System.out.println(kuyruk.poll());
9 System.out.println(kuyruk.size());
10 System.out.println(kuyruk.poll());
11 System.out.println(kuyruk.poll());
```

Banu

çıkış

Cem

← giriş

baş = son

Sıradaki Banu çıkar.

Adım 8 / 11

ÇIKTI

[Ali, Banu, Cem]

Ali

Ali

Banu

Queue: İzleme

```
1 Queue<String> kuyruk = new LinkedList<>();
2 kuyruk.offer("Ali");
3 kuyruk.offer("Banu");
4 kuyruk.offer("Cem");
5 System.out.println(kuyruk);
6 System.out.println(kuyruk.peek());
7 System.out.println(kuyruk.poll());
8 System.out.println(kuyruk.poll());
9 ► System.out.println(kuyruk.size());
10 System.out.println(kuyruk.poll());
11 System.out.println(kuyruk.poll());
```

çıkış

Cem

← giriş

baş = son**size()** : kuyrukta 1 öge kaldı.

Adım 9 / 11

ÇIKTI

```
[Ali, Banu, Cem]
```

```
Ali
```

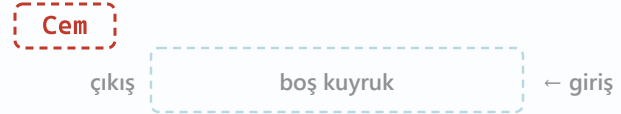
```
Ali
```

```
Banu
```

```
1
```

Queue: İzleme

```
1 Queue<String> kuyruk = new LinkedList<>();
2 kuyruk.offer("Ali");
3 kuyruk.offer("Banu");
4 kuyruk.offer("Cem");
5 System.out.println(kuyruk);
6 System.out.println(kuyruk.peek());
7 System.out.println(kuyruk.poll());
8 System.out.println(kuyruk.poll());
9 System.out.println(kuyruk.size());
10 System.out.println(kuyruk.poll());
11 System.out.println(kuyruk.poll());
```



Cem de çıkar; kuyruk boşaldı.

Adım 10 / 11

ÇIKTI

```
[Ali, Banu, Cem]
```

```
Ali
```

```
Ali
```

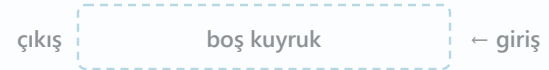
```
Banu
```

```
1
```

```
Cem
```

Queue: İzleme

```
1 Queue<String> kuyruk = new LinkedList<>();
2 kuyruk.offer("Ali");
3 kuyruk.offer("Banu");
4 kuyruk.offer("Cem");
5 System.out.println(kuyruk);
6 System.out.println(kuyruk.peek());
7 System.out.println(kuyruk.poll());
8 System.out.println(kuyruk.poll());
9 System.out.println(kuyruk.size());
10 System.out.println(kuyruk.poll());
▶ System.out.println(kuyruk.poll());
```



Boş kuyrukta `poll()` hata vermez, `null` döndürür.

Adım 11 / 11

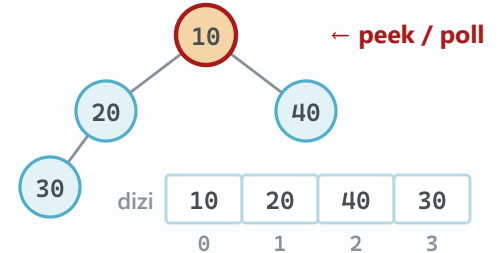
ÇIKTI

```
[Ali, Banu, Cem]
Ali
Ali
Banu
1
Cem
null
```

PriorityQueue Sınıfı

- **Öncelikli kuyruk** (priority queue): çıkan öge her zaman **en öncelikli** olandır, eklenme sırası değil.
- Varsayılan öncelik doğal sıradır: en küçük öge önce çıkar. Tersini için karşılaştırıcı verilir.
- Öğeler karşılaştırılabilir (`Comparable`) olmalı ya da bir karşılaştırıcı (`Comparator`) verilmelidir.
- İçte **ikili yığın** (binary heap) kullanır: `add` / `poll` $O(\log n)$, `peek` $O(1)$; dinamik olarak büyür.
- Kullanım: görev zamanlama, acil servis sırası, en kısa yol algoritmaları.

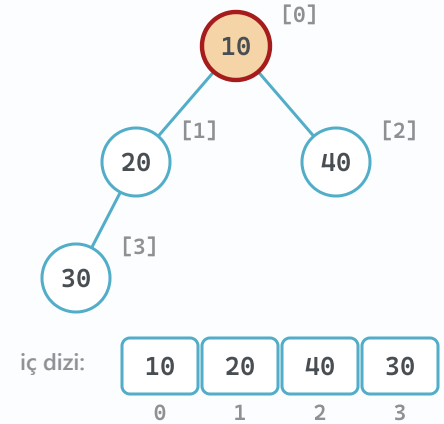
Metot	Görevi
<code>add(E e)</code>	öge ekler
<code>poll()</code>	en öncelikliyi çıkarır ve döndürür
<code>peek()</code>	en öncelikliyi döndürür, çıkarmaz
<code>remove(Object o)</code>	belirtilen öğeyi çıkarır
<code>size()</code>	öge sayısı



PriorityQueue Örneği

```
PriorityQueue<Integer> pq = new PriorityQueue<>();
pq.add(30);
pq.add(10);
pq.add(40);
pq.add(20);
System.out.println(pq.peek()); // en küçük
System.out.println(pq);       // iç düzen (yığın)
while (!pq.isEmpty()) {
    System.out.print(pq.poll() + " ");
}
System.out.println();
```

```
10
[10, 20, 40, 30]
10 20 30 40
```



`println(pq)` sıralı **değildir**:
iç dizi yalnızca “her ebeveyn
çocuklarından küçük” kuralına
uyar. Sıralı sonuç için `poll`
ile tek tek çıkarılır.

PriorityQueue: Yığın Kuralı Adım Adım

```
► PriorityQueue<Integer> pq = new PriorityQueue<>();
2 pq.add(30);
3 pq.add(10);
4 pq.add(40);
5 pq.add(20);
6 System.out.println(pq.peek());
7 System.out.println(pq);
8 while (!pq.isEmpty()) {
9     System.out.print(pq.poll() + " ");
10 }
11 System.out.println();
```

(boş)

dizi



İçte boş bir dizi: **ikili yığın** (binary heap). Kural: her ebeveyn çocuklarından küçük ya da eşit.

Adım 1 / 12

DEĞİŞKENLER

—

ÇIKTI

PriorityQueue: Yığın Kuralı Adım Adım

```
1 PriorityQueue<Integer> pq = new PriorityQueue<>();
2 ▶ pq.add(30);
3 pq.add(10);
4 pq.add(40);
5 pq.add(20);
6 System.out.println(pq.peek());
7 System.out.println(pq);
8 while (!pq.isEmpty()) {
9     System.out.print(pq.poll() + " ");
10 }
11 System.out.println();
```

30

dizi



30 köke (indis 0) yerleşir.

Adım 2 / 12

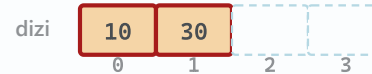
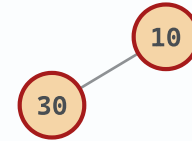
DEĞİŞKENLER

—

ÇIKTI

PriorityQueue: Yığın Kuralı Adım Adım

```
1 PriorityQueue<Integer> pq = new PriorityQueue<>();
2 pq.add(30);
3 ► pq.add(10);
4 pq.add(40);
5 pq.add(20);
6 System.out.println(pq.peek());
7 System.out.println(pq);
8 while (!pq.isEmpty()) {
9     System.out.print(pq.poll() + " ");
10 }
11 System.out.println();
```



10 sona (indis 1) eklenir; ebeveyni 30'dan küçük olduğu için **yukarı çıkar**, yer değiştirirler. Kök artık 10.

Adım 3 / 12

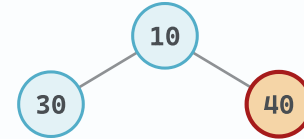
DEĞİŞKENLER

—

ÇIKTI

PriorityQueue: Yığın Kuralı Adım Adım

```
1 PriorityQueue<Integer> pq = new PriorityQueue<>();
2 pq.add(30);
3 pq.add(10);
4 ► pq.add(40);
5 pq.add(20);
6 System.out.println(pq.peek());
7 System.out.println(pq);
8 while (!pq.isEmpty()) {
9     System.out.print(pq.poll() + " ");
10 }
11 System.out.println();
```



40 indis 2'ye; ebeveyni 10'dan büyük, yerinde kalır.

Adım 4 / 12

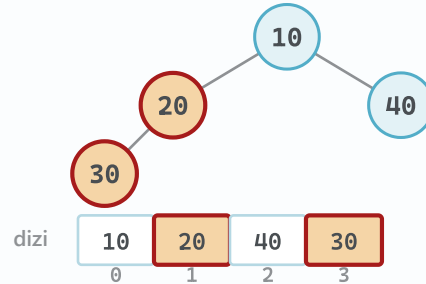
DEĞİŞKENLER

—

ÇIKTI

PriorityQueue: Yığın Kuralı Adım Adım

```
1 PriorityQueue<Integer> pq = new PriorityQueue<>();
2 pq.add(30);
3 pq.add(10);
4 pq.add(40);
5 ► pq.add(20);
6 System.out.println(pq.peek());
7 System.out.println(pq);
8 while (!pq.isEmpty()) {
9     System.out.print(pq.poll() + " ");
10 }
11 System.out.println();
```



20 indis 3'e; ebeveyni 30'dan küçük → yer değiştirir; 10'dan büyük → durur.

Adım 5 / 12

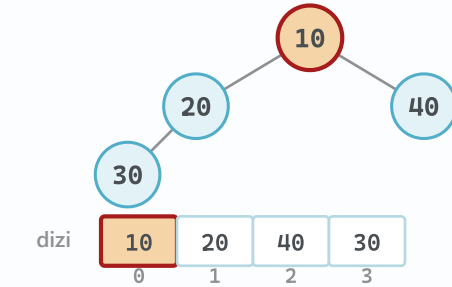
DEĞİŞKENLER

—

ÇIKTI

PriorityQueue: Yığın Kuralı Adım Adım

```
1 PriorityQueue<Integer> pq = new PriorityQueue<>();
2 pq.add(30);
3 pq.add(10);
4 pq.add(40);
5 pq.add(20);
6 ► System.out.println(pq.peek());
7 System.out.println(pq);
8 while (!pq.isEmpty()) {
9     System.out.print(pq.poll() + " ");
10 }
11 System.out.println();
```



`peek()` : kök, yani en küçük öge: 10. Kuyruk değişmez; $O(1)$.

Adım 6 / 12

DEĞİŞKENLER

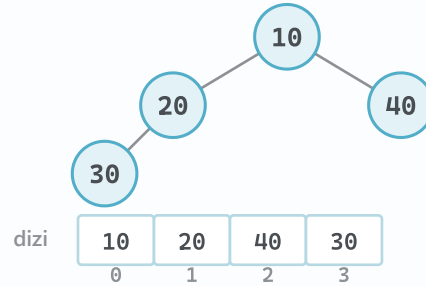
—

ÇIKTI

10

PriorityQueue: Yığın Kuralı Adım Adım

```
1 PriorityQueue<Integer> pq = new PriorityQueue<>();
2 pq.add(30);
3 pq.add(10);
4 pq.add(40);
5 pq.add(20);
6 System.out.println(pq.peek());
7 ► System.out.println(pq);
8 while (!pq.isEmpty()) {
9     System.out.print(pq.poll() + " ");
10 }
11 System.out.println();
```



println iç diziyi olduğu gibi yazar: [10, 20, 40, 30] sıralı değildir.

Adım 7 / 12

DEĞİŞKENLER

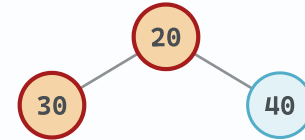
—

ÇIKTI

```
10
[10, 20, 40, 30]
```

PriorityQueue: Yığın Kuralı Adım Adım

```
1 PriorityQueue<Integer> pq = new PriorityQueue<>();
2 pq.add(30);
3 pq.add(10);
4 pq.add(40);
5 pq.add(20);
6 System.out.println(pq.peek());
7 System.out.println(pq);
8 while (!pq.isEmpty()) {
9     System.out.print(pq.poll() + " ");
10 }
11 System.out.println();
```



`poll()` : kök 10 çıkar; son öge 30 köke gelir ve küçük çocuğu 20 ile yer değiştirip **aşağı iner**.

Adım 8 / 12

DEĞİŞKENLER

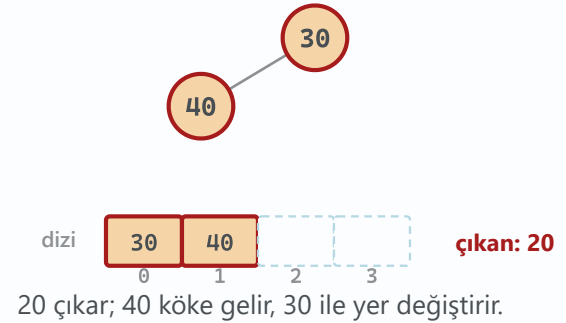
x = 10

ÇIKTI

```
10
[10, 20, 40, 30]
10
```

PriorityQueue: Yığın Kuralı Adım Adım

```
1 PriorityQueue<Integer> pq = new PriorityQueue<>();
2 pq.add(30);
3 pq.add(10);
4 pq.add(40);
5 pq.add(20);
6 System.out.println(pq.peek());
7 System.out.println(pq);
8 while (!pq.isEmpty()) {
9     System.out.print(pq.poll() + " ");
10 }
11 System.out.println();
```



Adım 9 / 12

DEĞİŞKENLER

x = 20

ÇIKTI

```
10
[10, 20, 40, 30]
10 20
```

PriorityQueue: Yığın Kuralı Adım Adım

```
1 PriorityQueue<Integer> pq = new PriorityQueue<>();
2 pq.add(30);
3 pq.add(10);
4 pq.add(40);
5 pq.add(20);
6 System.out.println(pq.peek());
7 System.out.println(pq);
8 while (!pq.isEmpty()) {
9     System.out.print(pq.poll() + " ");
10 }
11 System.out.println();
```

40

dizi



çıkan: 30

30 çıkar; tek öge 40 kaldı.

Adım 10 / 12

DEĞİŞKENLER

x = 30

ÇIKTI

```
10
[10, 20, 40, 30]
10 20 30
```

PriorityQueue: Yığın Kuralı Adım Adım

```
1 PriorityQueue<Integer> pq = new PriorityQueue<>();
2 pq.add(30);
3 pq.add(10);
4 pq.add(40);
5 pq.add(20);
6 System.out.println(pq.peek());
7 System.out.println(pq);
8 while (!pq.isEmpty()) {
9     System.out.print(pq.poll() + " ");
10 }
11 System.out.println();
```

(boş)

dizi



çıkan: 40

40 çıkar; kuyruk boş, döngü biter. Çıkış sırası sıralı:
10 20 30 40.

Adım 11 / 12

DEĞİŞKENLER

x = 40

ÇIKTI

```
10
[10, 20, 40, 30]
10 20 30 40
```

PriorityQueue: Yığın Kuralı Adım Adım

```
1 PriorityQueue<Integer> pq = new PriorityQueue<>();
2 pq.add(30);
3 pq.add(10);
4 pq.add(40);
5 pq.add(20);
6 System.out.println(pq.peek());
7 System.out.println(pq);
8 while (!pq.isEmpty()) {
9     System.out.print(pq.poll() + " ");
10 }
▶ System.out.println();
```

(boş)

dizi



Satır sonu yazılır. Her `poll` yalnızca bir dalı izler: $O(\log n)$.

Adım 12 / 12

DEĞİŞKENLER

—

ÇIKTI

```
10
[10, 20, 40, 30]
10 20 30 40
```

Önceliği Belirlemek

```
class Gorev implements Comparable<Gorev> {
    String ad;
    int oncelik;           // küçük = acil

    Gorev(String ad, int oncelik) {
        this.ad = ad;
        this.oncelik = oncelik;
    }

    @Override
    public int compareTo(Gorev diger) {
        return Integer.compare(oncelik, diger.oncelik);
    }

    @Override
    public String toString() {
        return ad + "(" + oncelik + ")";
    }
}
```

`compareTo` negatif dönerse `this` önce gelir.

`Collections.reverseOrder()` doğal sırayı tersine çevirir: en büyük önce çıkar.

```
PriorityQueue<Gorev> gorevler =
    new PriorityQueue<>();
gorevler.add(new Gorev("Rapor", 3));
gorevler.add(new Gorev("Arıza", 1));
gorevler.add(new Gorev("Toplantı", 2));
while (!gorevler.isEmpty()) {
    System.out.println(gorevler.poll());
}
```

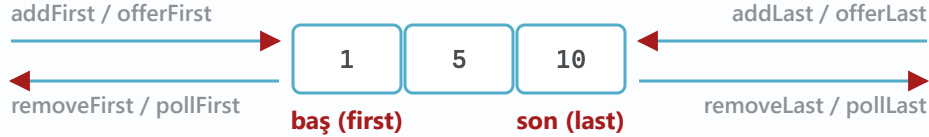
```
Arıza(1)
Toplantı(2)
Rapor(3)
```

```
PriorityQueue<Integer> buyukOnce =
    new PriorityQueue<>(
        Collections.reverseOrder());
buyukOnce.add(30);
buyukOnce.add(10);
buyukOnce.add(40);
System.out.println(buyukOnce.poll());
```

KUYRUKLAR

Deque Arayüzü

- **Deque** (Double-Ended Queue, "dek" okunur): hem baştan hem sondan öge eklenebilen ve çıkarılabilen **çift uçlu kuyruk**.
- **Queue** (FIFO) ve **Stack** (LIFO) işlevselliğini birleştirir.
- Uygulamalar: **LinkedList** (çift yönlü bağlı liste) ve **ArrayDeque** (dinamik dizi).



Baş	Görevi	Son	Görevi
<code>addFirst(e)</code>	başa ekler	<code>addLast(e)</code>	sona ekler
<code>removeFirst()</code>	baştakini çıkarır, döndürür	<code>removeLast()</code>	sondakini çıkarır, döndürür
<code>getFirst()</code>	baştakini döndürür	<code>getLast()</code>	sondakini döndürür

`remove...` / `get...` boş deque'de istisna fırlatır; `poll...` / `peek...` `null` döndürür.
`size()` : öge sayısı.

ArrayDeque Sınıfı

- Çift uçlu kuyruk yapısını **dinamik (dairesel) bir diziyle** gerçekleştirir.
- Hem kuyruk hem yığın işlemlerini destekler.
- Genellikle `LinkedList` 'ten ve `Stack` 'ten daha hızlı ve hafiftir; yığın ve kuyruk için önerilen sınıftır.
- `null` öğe kabul etmez.

Metot	Görevi
<code>pollFirst()</code> / <code>pollLast()</code>	baştakini / sondakini çıkarır, döndürür
<code>peekFirst()</code> / <code>peekLast()</code>	baştakini / sondakini döndürür, çıkarmaz

YIĞIN OLARAK (STACK YERİNE)

```
Deque<String> yigin = new ArrayDeque<>();
yigin.push("A");      // başa ekler
yigin.push("B");
System.out.println(yigin.pop());
```

B

`push` / `pop` / `peek` baş uçta çalışır.

Kuyruk olarak: `offer` sona ekler, `poll` baştan alır.



ArrayDeque: İzleme

```
► ArrayDeque<Integer> sayilar = new ArrayDeque<>();  
2 sayilar.addFirst(5);  
3 sayilar.addLast(10);  
4 sayilar.addFirst(1);  
5 System.out.println(sayilar);  
6 int ilkSayi = sayilar.pollFirst();  
7 int sonSayi = sayilar.pollLast();  
8 System.out.println(ilkSayi + " " + sonSayi);  
9 System.out.println(sayilar.peekFirst());
```

boş

Boş bir `ArrayDeque` .

Adım 1 / 9

DEĞİŞKENLER

—

ÇIKTI

ArrayDeque: İzleme

```
1 ArrayDeque<Integer> sayilar = new ArrayDeque<>();  
2 sayilar.addFirst(5);  
3 sayilar.addLast(10);  
4 sayilar.addFirst(1);  
5 System.out.println(sayilar);  
6 int ilkSayi = sayilar.pollFirst();  
7 int sonSayi = sayilar.pollLast();  
8 System.out.println(ilkSayi + " " + sonSayi);  
9 System.out.println(sayilar.peekFirst());
```

5

baş = son

addFirst(5) : 5 başa eklenir (tek öge).

Adım 2 / 9

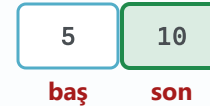
DEĞİŞKENLER

—

ÇIKTI

ArrayDeque: İzleme

```
1 ArrayDeque<Integer> sayilar = new ArrayDeque<>();
2 sayilar.addFirst(5);
3 ► sayilar.addLast(10);
4 sayilar.addFirst(1);
5 System.out.println(sayilar);
6 int ilkSayi = sayilar.pollFirst();
7 int sonSayi = sayilar.pollLast();
8 System.out.println(ilkSayi + " " + sonSayi);
9 System.out.println(sayilar.peekFirst());
```



`addLast(10)` : 10 sona eklenir.

Adım 3 / 9

DEĞİŞKENLER

—

ÇIKTI

ArrayDeque: İzleme

```
1 ArrayDeque<Integer> sayilar = new ArrayDeque<>();
2 sayilar.addFirst(5);
3 sayilar.addLast(10);
4 ► sayilar.addFirst(1);
5 System.out.println(sayilar);
6 int ilkSayi = sayilar.pollFirst();
7 int sonSayi = sayilar.pollLast();
8 System.out.println(ilkSayi + " " + sonSayi);
9 System.out.println(sayilar.peekFirst());
```



`addFirst(1)` : 1 başa eklenir, 5 ve 10 yerinde kalır.

Adım 4 / 9

DEĞİŞKENLER

—

ÇIKTI

ArrayDeque: İzleme

```
1 ArrayDeque<Integer> sayilar = new ArrayDeque<>();
2 sayilar.addFirst(5);
3 sayilar.addLast(10);
4 sayilar.addFirst(1);
5 System.out.println(sayilar);
6 int ilkSayi = sayilar.pollFirst();
7 int sonSayi = sayilar.pollLast();
8 System.out.println(ilkSayi + " " + sonSayi);
9 System.out.println(sayilar.peekFirst());
```



Baştan sona yazdırılır.

Adım 5 / 9

DEĞİŞKENLER

—

ÇIKTI

[1, 5, 10]

ArrayDeque: İzleme

```
1 ArrayDeque<Integer> sayilar = new ArrayDeque<>();
2 sayilar.addFirst(5);
3 sayilar.addLast(10);
4 sayilar.addFirst(1);
5 System.out.println(sayilar);
▶ int ilkSayi = sayilar.pollFirst();
7 int sonSayi = sayilar.pollLast();
8 System.out.println(ilkSayi + " " + sonSayi);
9 System.out.println(sayilar.peekFirst());
```



`pollFirst()` baştaki 1'i çıkarır; otomatik kutu açma ile `int` 'e atanır.

Adım 6 / 9

DEĞİŞKENLER

`ilkSayi = 1`

ÇIKTI

`[1, 5, 10]`

ArrayDeque: İzleme

```
1 ArrayDeque<Integer> sayilar = new ArrayDeque<>();
2 sayilar.addFirst(5);
3 sayilar.addLast(10);
4 sayilar.addFirst(1);
5 System.out.println(sayilar);
6 int ilkSayi = sayilar.pollFirst();
7 int sonSayi = sayilar.pollLast();
8 System.out.println(ilkSayi + " " + sonSayi);
9 System.out.println(sayilar.peekFirst());
```



baş = son

`pollLast()` sondaki 10'u çıkarır.

Adım 7 / 9

DEĞİŞKENLER

`ilkSayi = 1`

`sonSayi = 10`

ÇIKTI

`[1, 5, 10]`

ArrayDeque: İzleme

```
1 ArrayDeque<Integer> sayilar = new ArrayDeque<>();
2 sayilar.addFirst(5);
3 sayilar.addLast(10);
4 sayilar.addFirst(1);
5 System.out.println(sayilar);
6 int ilkSayi = sayilar.pollFirst();
7 int sonSayi = sayilar.pollLast();
▶ System.out.println(ilkSayi + " " + sonSayi);
9 System.out.println(sayilar.peekFirst());
```

5

baş = son

İki değer yazdırılır.

Adım 8 / 9

DEĞİŞKENLER

ilkSayi = 1

sonSayi = 10

ÇIKTI

[1, 5, 10]

1 10

ArrayDeque: İzleme

```
1 ArrayDeque<Integer> sayilar = new ArrayDeque<>();
2 sayilar.addFirst(5);
3 sayilar.addLast(10);
4 sayilar.addFirst(1);
5 System.out.println(sayilar);
6 int ilkSayi = sayilar.pollFirst();
7 int sonSayi = sayilar.pollLast();
8 System.out.println(ilkSayi + " " + sonSayi);
▶ System.out.println(sayilar.peekFirst());
```

5

baş = son

`peekFirst()` baştaki 5'i döndürür, çıkarmaz.

Adım 9 / 9

DEĞİŞKENLER

ilkSayi = 1

sonSayi = 10

ÇIKTI

[1, 5, 10]

1 10

5

Set Arayüzü

- **Tekil** öğeleri saklar; matematikteki küme gibidir ve küme işlemlerine uygundur.
- Her öğeden yalnızca bir tane bulunur: var olan öğeyi `add` eklemeyiz, `false` döndürür.
- Genel olarak sıra garantisi vermez (sıra, uygulamaya bağlıdır).
- Ekleme, çıkarma ve arama hızlıdır.
- `HashSet` : sırasız. `LinkedHashSet` : eklenme sırası. `TreeSet` : sıralı (doğal sıra ya da `Comparator`).

Metot	Görevi
<code>add(E e)</code>	öğeyi kümeye ekler
<code>remove(Object o)</code>	öğeyi kümeden çıkarır
<code>contains(Object o)</code>	öğ kümede var mı?
<code>size()</code>	öğ sayısı

```
Set<String> meyveler = new HashSet<>();
System.out.println(meyveler.add("Elma"));
System.out.println(meyveler.add("Armut"));
System.out.println(meyveler.add("Elma"));
System.out.println(meyveler.size());
System.out.println(meyveler.contains("Armut"));
meyveler.remove("Armut");
System.out.println(meyveler);
```

```
true
true
false
2
true
[Elma]
```

HashSet ve Hashing

- Tekil öğeleri **sirasız** saklar; `AbstractSet` 'ten türetilmiştir ve `Set` 'i gerçekleştirir.
- Öğenin `hashCode()` değerinden bir **kova** (bucket) numarası hesaplanır; öğe doğrudan o kovaya gider: ekleme ve arama ortalama $O(1)$.
- Depolama sırası garanti edilmez: gezinme kovaların sırasıyla yapılır.



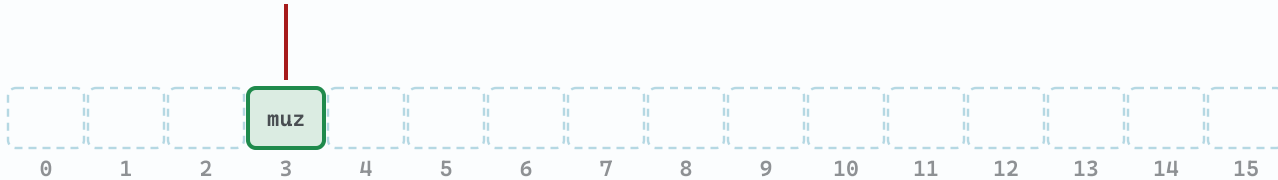
kovalar (bucket): 0 ... 15

İşte 16 kovalı bir tablo vardır (dolulukla birlikte büyür).

HashSet ve Hashing

- Tekil öğeleri **sirasız** saklar; `AbstractSet` 'ten türetilmiştir ve `Set` 'i gerçekleştirir.
- Öğenin `hashCode()` değerinden bir **kova** (bucket) numarası hesaplanır; öğe doğrudan o kovaya gider: ekleme ve arama ortalama $O(1)$.
- Depolama sırası garanti edilmez: gezinme kovaların sırasıyla yapılır.

"muz".hashCode() = 108498 → kova 3



kovalar (bucket): 0 ... 15

`add("muz")` : hash değerinden kova 3 hesaplanır, muz oraya yerleşir.

HashSet ve Hashing

- Tekil öğeleri **sirasız** saklar; `AbstractSet` 'ten türetilmiştir ve `Set` 'i gerçekleştirir.
- Öğenin `hashCode()` değerinden bir **kova** (bucket) numarası hesaplanır; öğe doğrudan o kovaya gider: ekleme ve arama ortalama $O(1)$.
- Depolama sırası garanti edilmez: gezinme kovaların sırasıyla yapılır.

"elma".hashCode() = 3116155 → kova 4



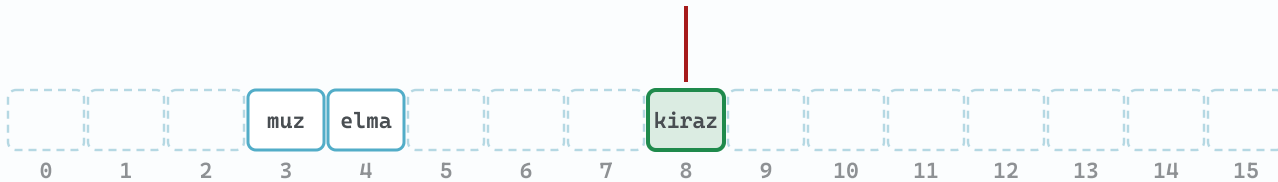
kovalar (bucket): 0 ... 15

`add("elma")` : kova 4.

HashSet ve Hashing

- Tekil öğeleri **sirasız** saklar; `AbstractSet` 'ten türetilmiştir ve `Set` 'i gerçekleştirir.
- Öğenin `hashCode()` değerinden bir **kova** (bucket) numarası hesaplanır; öğe doğrudan o kovaya gider: ekleme ve arama ortalama $O(1)$.
- Depolama sırası garanti edilmez: gezinme kovaların sırasıyla yapılır.

"kiraz".hashCode() = 102057485 → kova 8



kovalar (bucket): 0 ... 15

`add("kiraz")` : kova 8.

HashSet ve Hashing

- Tekil öğeleri **sirasız** saklar; `AbstractSet` 'ten türetilmiştir ve `Set` 'i gerçekleştirir.
- Öğenin `hashCode()` değerinden bir **kova** (bucket) numarası hesaplanır; öğe doğrudan o kovaya gider: ekleme ve arama ortalama $O(1)$.
- Depolama sırası garanti edilmez: gezinme kovaların sırasıyla yapılır.

"elma".hashCode() = 3116155 → kova 4

zaten var → eklenmez



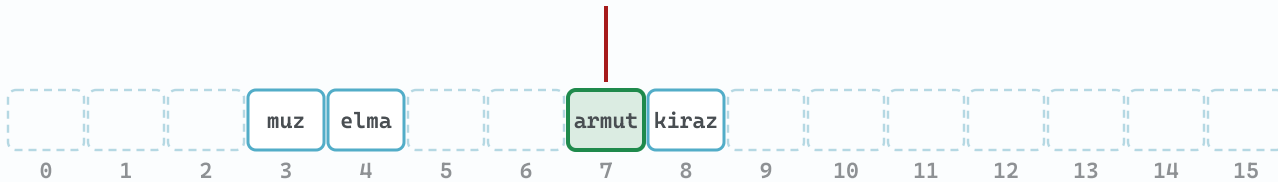
kovalar (bucket): 0 ... 15

İkinci `add("elma")` : kova 4'te `equals` ile eşit bir öğe var. Eklenmez; `add` `false` döner.

HashSet ve Hashing

- Tekil öğeleri **sirasız** saklar; `AbstractSet` 'ten türetilmiştir ve `Set` 'i gerçekleştirir.
- Öğenin `hashCode()` değerinden bir **kova** (bucket) numarası hesaplanır; öğe doğrudan o kovaya gider: ekleme ve arama ortalama $O(1)$.
- Depolama sırası garanti edilmez: gezinme kovaların sırasıyla yapılır.

"armut".hashCode() = 93086203 → kova 7



kovalar (bucket): 0 ... 15

`add("armut")` : kova 7. Gezinme sırası kovalara göre: `[muz, elma, armut, kiraz]` .

LinkedHashSet Sınıfı

- Sadece tekil öğeleri içeren bir küme;
`HashSet` 'ten türer.
- Ek olarak öğeleri bir bağlı listeye birbirine bağlar:
eklenme sırası korunur.
- Hızlı erişim sağlar ve öğelerin benzersizliğini garanti eder.
- Hem performans hem sıra önemliyse tercih edilir; sıra önemli değilse ve performans kritikse `HashSet` kullanılır.

```
LinkedHashSet<Integer> sayilar =  
    new LinkedHashSet<>();  
sayilar.add(5);  
sayilar.add(10);  
sayilar.add(5);           // tekrar: eklenmez  
sayilar.add(1);  
System.out.println(sayilar);
```

[5, 10, 1]

Metot	Görevi
<code>add(E e)</code>	öge ekler
<code>remove(Object o)</code>	ögeyi çıkarır
<code>contains(Object o)</code>	öge var mı?
<code>size()</code>	öge sayısı
<code>clear()</code>	tüm öğeleri çıkarır

KÜMELER

Üç Kümenin Sırası

```
String[] kelimeler = {"muz", "elma", "kiraz", "elma", "armut"};
Set<String> h = new HashSet<>();
Set<String> l = new LinkedHashSet<>();
Set<String> t = new TreeSet<>();
for (String k : kelimeler) {
    h.add(k);
    l.add(k);
    t.add(k);
}
System.out.println("HashSet:      " + h);
System.out.println("LinkedHashSet: " + l);
System.out.println("TreeSet:      " + t);
```

```
HashSet:      [muz, elma, armut, kiraz]
LinkedHashSet: [muz, elma, kiraz, armut]
TreeSet:      [armut, elma, kiraz, muz]
```

Sınıf	Gezinme sırası	Temel yapı	add / contains
HashSet	garanti yok (kova sırası)	hash tablosu	O(1) ortalama
LinkedHashSet	eklenme sırası	hash tablosu + bağlı liste	O(1) ortalama
TreeSet	sıralı (küçükten büyüğe)	kırmızı-siyah ağaç	O(log n)

Kendi Sınıflarımız ve HashSet

```
class Nokta {  
    int x, y;  
    Nokta(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
    @Override  
    public boolean equals(Object o) {  
        if (!(o instanceof Nokta)) return false;  
        Nokta n = (Nokta) o;  
        return x == n.x && y == n.y;  
    }  
    @Override  
    public int hashCode() {  
        return Objects.hash(x, y);  
    }  
}
```

NOKTAA: AYNI SINIF, EQUALS/HASHCODE YOK

```
Set<NoktaA> s1 = new HashSet<>();  
s1.add(new NoktaA(1, 2));  
s1.add(new NoktaA(1, 2));  
System.out.println(s1.size());
```

```
Set<Nokta> s2 = new HashSet<>();  
s2.add(new Nokta(1, 2));  
s2.add(new Nokta(1, 2));  
System.out.println(s2.size());
```

2
1

- **NoktaA** : iki nesne **farklı** sayılır (Object'in **equals** 'ı referansları karşılaştırır) → boyut 2.
- **Nokta** : **equals** ve **hashCode** geçersiz kılındı → aynı nokta bir kez → boyut 1.

Kural: eşit nesnelerin **hashCode** değeri de eşit olmalı: kova **hashCode** ile bulunur, sonra **equals** ile karşılaştırılır.

Kümedeki Nesneyi Değiştirmek: Kayıp Öğe

```
► Set<Nokta> kume = new HashSet<>();
2 Nokta p = new Nokta(1, 2);
3 kume.add(p);
4 System.out.println(kume.contains(p));
5 p.x = 5;           // alan değişti
6 System.out.println(kume.contains(p));
7 System.out.println(kume.contains(new Nokta(1, 2)));
8 kume.add(p);
9 System.out.println(kume.size());
```

kovalar

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
---	---	---	---	---	---	---	---	---	---	----	----	----	----	----	----

`HashSet` içte 16 kovalı bir hash tablosu kullanır. `Nokta` 'nın `equals` ve `hashCode` metotları `x` ve `y` 'den hesaplanır.

Adım 1 / 9

ÇIKTI



Kümedeki Nesneyi Değiştirmek: Kayıp Öğe

```
1 Set<Nokta> kume = new HashSet<>();  
▶ Nokta p = new Nokta(1, 2);  
3 kume.add(p);  
4 System.out.println(kume.contains(p));  
5 p.x = 5; // alan değişti  
6 System.out.println(kume.contains(p));  
7 System.out.println(kume.contains(new Nokta(1, 2)));  
8 kume.add(p);  
9 System.out.println(kume.size());
```

kovalar

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
---	---	---	---	---	---	---	---	---	---	----	----	----	----	----	----

p

(1, 2)

hash = 994 → kova 2

`Objects.hash(1, 2)` = 994. Kova
numarası hash'in son 4 bitidir: 994 → kova 2.

Adım 2 / 9

ÇIKTI



Kümedeki Nesneyi Değiştirmek: Kayıp Öğe

```
1 Set<Nokta> kume = new HashSet<>();
2 Nokta p = new Nokta(1, 2);
▶ kume.add(p);
4 System.out.println(kume.contains(p));
5 p.x = 5; // alan değişti
6 System.out.println(kume.contains(p));
7 System.out.println(kume.contains(new Nokta(1, 2)));
8 kume.add(p);
9 System.out.println(kume.size());
```

kovalar



p

(1, 2)

hash = 994 → kova 2

add(p) : nesne 2 numaralı kovaya yerleşir.

Adım 3 / 9

ÇIKTI



Kümedeki Nesneyi Değiştirmek: Kayıp Öğe

```
1 Set<Nokta> kume = new HashSet<>();
2 Nokta p = new Nokta(1, 2);
3 kume.add(p);
4 ► System.out.println(kume.contains(p));
5 p.x = 5; // alan değişti
6 System.out.println(kume.contains(p));
7 System.out.println(kume.contains(new Nokta(1, 2)));
8 kume.add(p);
9 System.out.println(kume.size());
```

kovalar



p

(1, 2)

hash = 994 → kova 2

`contains(p)` : yine kova 2'ye bakılır,
`equals` doğru → `true` .

Adım 4 / 9

ÇIKTI

true

Kümedeki Nesneyi Değiştirmek: Kayıp Öğe

```
1 Set<Nokta> kume = new HashSet<>();
2 Nokta p = new Nokta(1, 2);
3 kume.add(p);
4 System.out.println(kume.contains(p));
5 ► p.x = 5; // alan değişti
6 System.out.println(kume.contains(p));
7 System.out.println(kume.contains(new Nokta(1, 2)));
8 kume.add(p);
9 System.out.println(kume.size());
```

kovalar

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
---	---	---	---	---	---	---	---	---	---	----	----	----	----	----	----

p

(5, 2)

hash = 1118 → kova 14

Kümedeki nesnenin alanı değişti! hashCode artık 1118 → kova 14. Ama nesne hâlâ **kova 2'de** duruyor.

Adım 5 / 9

ÇIKTI

true

Kümedeki Nesneyi Değiştirmek: Kayıp Öğe

```
1 Set<Nokta> kume = new HashSet<>();
2 Nokta p = new Nokta(1, 2);
3 kume.add(p);
4 System.out.println(kume.contains(p));
5 p.x = 5; // alan değişti
6 ► System.out.println(kume.contains(p));
7 System.out.println(kume.contains(new Nokta(1, 2)));
8 kume.add(p);
9 System.out.println(kume.size());
```

kovalar

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
---	---	---	---	---	---	---	---	---	---	----	----	----	----	----	----

p

(5, 2)

hash = 1118 → kova 14

`contains(p)` : kova 14'e bakılır; orası boş
→ `false` . Nesne kümede, ama bulunamıyor.

Adım 6 / 9

ÇIKTI

```
true
false
```

Kümedeki Nesneyi Değiştirmek: Kayıp Öğe

```
1 Set<Nokta> kume = new HashSet<>();
2 Nokta p = new Nokta(1, 2);
3 kume.add(p);
4 System.out.println(kume.contains(p));
5 p.x = 5; // alan değişti
6 System.out.println(kume.contains(p));
7 ► System.out.println(kume.contains(new Nokta(1, 2)));
8 kume.add(p);
9 System.out.println(kume.size());
```

kovalar



p

(5, 2)

hash = 1118 → kova 14

Yeni `Nokta(1, 2)` kova 2'ye gider;
oradaki nesne artık (5, 2), `equals` yanlış
→ `false`.

Adım 7 / 9

ÇIKTI

```
true
false
false
```

Kümedeki Nesneyi Değiştirmek: Kayıp Öğe

```
1 Set<Nokta> kume = new HashSet<>();
2 Nokta p = new Nokta(1, 2);
3 kume.add(p);
4 System.out.println(kume.contains(p));
5 p.x = 5; // alan değişti
6 System.out.println(kume.contains(p));
7 System.out.println(kume.contains(new Nokta(1, 2)));
8 ► kume.add(p);
9 System.out.println(kume.size());
```

kovalar



p

(5, 2)

hash = 1118 → kova 14

add(p) : kova 14 boş görüldüğü için **aynı nesne ikinci kez** eklenir.

Adım 8 / 9

ÇIKTI

```
true
false
false
```

Kümedeki Nesneyi Değiştirmek: Kayıp Öğe

```
1 Set<Nokta> kume = new HashSet<>();
2 Nokta p = new Nokta(1, 2);
3 kume.add(p);
4 System.out.println(kume.contains(p));
5 p.x = 5;           // alan değişti
6 System.out.println(kume.contains(p));
7 System.out.println(kume.contains(new Nokta(1, 2)));
8 kume.add(p);
▶ System.out.println(kume.size());
```

kovalar



p •

(5, 2)

hash = 1118 → kova 14

`size()` = 2: küme bozuldu. Hash'li koleksiyondaki nesnenin equals/hashCode alanları değiştirilmez; anahtar olarak `String`, `Integer` gibi değiştirilemez nesneler seçilir.

Adım 9 / 9

ÇIKTI

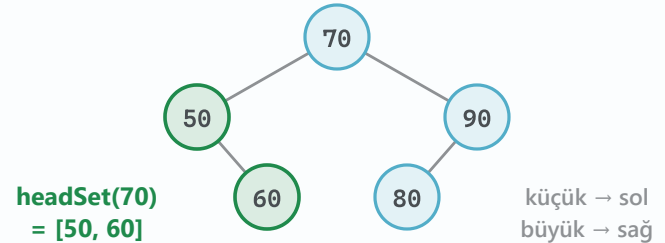
```
true
false
false
2
```

KÜMELER

SortedSet ve TreeSet

- **SortedSet** : öğeleri **sıralı** saklayan tekil küme arayüzü.
- Sıralama doğal sıraya (**Comparable**) ya da verilen bir **Comparator** 'a göre yapılır.
- Alt küme ve üst küme işlemleri sunar.
- **TreeSet** : **AbstractSet** 'ten türetilir, **NavigableSet** (dolayısıyla **SortedSet**) arayüzünü gerçekleştirir.
- İçte kırmızı-siyah ağaç: ekleme, silme, arama $O(\log n)$.

Metot	Görevi
<code>first()</code> / <code>last()</code>	en küçük / en büyük öğe
<code>headSet(to)</code>	<code>to</code> 'dan küçükler (<code>to</code> hariç)
<code>tailSet(from)</code>	<code>from</code> ve büyükler (<code>from</code> dahil)
<code>subSet(from, to)</code>	<code>from</code> dahil, <code>to</code> hariç
<code>comparator()</code>	sıralayan Comparator ; doğal sırada <code>null</code>
<code>add</code> , <code>remove</code>	ekleme, çıkarma



KÜMELER

TreeSet Örneği

```
TreeSet<Integer> notlar = new TreeSet<>();
notlar.add(70);
notlar.add(50);
notlar.add(90);
notlar.add(60);
notlar.add(80);
System.out.println(notlar);
System.out.println(notlar.first() + " " + notlar.last());
System.out.println(notlar.headSet(70)); // 70 hariç
System.out.println(notlar.tailSet(70)); // 70 dahil
System.out.println(notlar.subSet(60, 90)); // [60, 90)
System.out.println(notlar.comparator()); // doğal sıra
```

```
TreeSet<String> meyveler = new TreeSet<>();
meyveler.add("Elma");
meyveler.add("Armut");
System.out.println(meyveler);
```

ÇIKTI

```
[50, 60, 70, 80, 90]
50 90
[50, 60]
[70, 80, 90]
[60, 70, 80]
null
[Armut, Elma]
```

Öğeler hangi sırayla eklenirse eklensin sıralı tutulur. Dizgiler alfabetik (karakter kodu) sırayla dizilir.

KÜMELER

Küme İşlemleri

```
Set<Integer> a = new TreeSet<>(Arrays.asList(1, 2, 3, 4));  
Set<Integer> b = new TreeSet<>(Arrays.asList(3, 4, 5));
```

```
Set<Integer> birlesim = new TreeSet<>(a);  
birlesim.addAll(b);  
Set<Integer> kesisim = new TreeSet<>(a);  
kesisim.retainAll(b);  
Set<Integer> fark = new TreeSet<>(a);  
fark.removeAll(b);
```

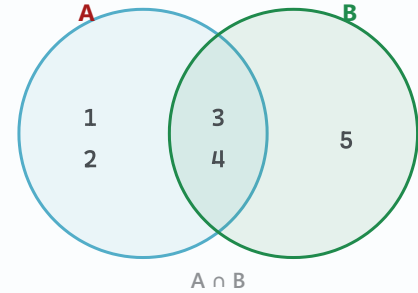
```
System.out.println("A ∪ B = " + birlesim);  
System.out.println("A ∩ B = " + kesisim);  
System.out.println("A - B = " + fark);  
System.out.println(a.containsAll(kesisim));
```

$A \cup B = [1, 2, 3, 4, 5]$

$A \cap B = [3, 4]$

$A - B = [1, 2]$

true



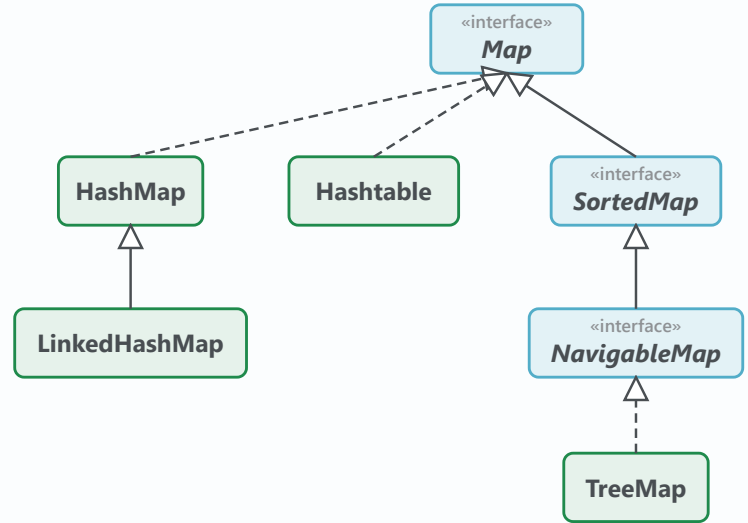
İşlem	Metot
birleşim $\cup$	<code>addAll</code>
kesişim $\cap$	<code>retainAll</code>
fark $-$	<code>removeAll</code>
alt küme $\subseteq$	<code>containsAll</code>

Bu metotlar çağrıldığı kümeyi **değiştirir**; bu yüzden önce kopya alınır.

Map Arayüzü

- **Anahtar-değer** (key-value) çiftlerini saklar: her anahtar bir değere eşlenir.
- Anahtarlar tekildir; değerler tekrar edebilir.
- Anahtar üzerinden hızlı erişim sağlar; çiftler üzerinde işlem yapmaya olanak tanır.
- **HashMap** : sıra garantisi yok.
LinkedHashMap : eklenme sırası.
TreeMap : anahtarlar sıralı (doğal sıra ya da **Comparator**).

Benzetme: sözlük. Kelime (anahtar) ile anlamını (değer) bulursunuz; aynı kelime iki kez yer almaz.



Map Metotları

Metot	Görevi
<code>put(K key, V value)</code>	anahtarı değerle eşler; anahtar varsa değeri değiştirir ve eskisini döndürür
<code>get(Object key)</code>	anahtarın değerini döndürür; anahtar yoksa <code>null</code>
<code>remove(Object key)</code>	anahtarı ve değerini çıkarır
<code>containsKey(Object key)</code>	anahtar eşlemede var mı?
<code>containsValue(Object v)</code>	değer eşlemede var mı?
<code>keySet()</code>	tüm anahtarları bir <code>Set</code> olarak döndürür
<code>values()</code>	tüm değerleri bir <code>Collection</code> olarak döndürür
<code>entrySet()</code>	tüm anahtar-değer çiftlerini bir <code>Set<Map.Entry<K, V>></code> olarak döndürür
<code>size()</code> , <code>isEmpty()</code>	çift sayısı, boş mu?
<code>getOrDefault(key, v)</code>	anahtar yoksa <code>v</code> 'yi döndürür (Java 8+)

`K` : anahtar tipi, `V` : değer tipi. Örnek: `Map<String, Integer>` isimden nota eşleme.

HashMap Sınıfı

```
HashMap<String, Double> urunFiyatlari = new HashMap<>();
urunFiyatlari.put("Telefon", 2000.0);
urunFiyatlari.put("Kulaklık", 350.0);
urunFiyatlari.put("Kılıf", 120.0);
Double eski = urunFiyatlari.put("Telefon", 1800.0);
System.out.println(eski);
System.out.println(urunFiyatlari.get("Telefon"));
System.out.println(urunFiyatlari.get("Tablet"));
System.out.println(urunFiyatlari.containsKey("Kılıf"));
urunFiyatlari.remove("Kılıf");
System.out.println(urunFiyatlari.size());
```

```
2000.0
1800.0
null
true
2
```

```
urunFiyatlari.put(null, 0.0);           // tek null anahtar
urunFiyatlari.put("Kalem", null);       // null değer
System.out.println(urunFiyatlari.size());
```

```
4
```

- Anahtar-değer çiftlerini **hash tablosunda** saklar: `get` / `put` ortalama $O(1)$.
- `AbstractMap` 'ten türetilmiştir ve `Map` 'i gerçekleştirir.
- **Tekil anahtarlar**: aynı anahtarla `put` değeri değiştirir.
- **Null**: bir `null` anahtara ve birden çok `null` değere izin verir.
- Öğeler sırasız depolanır.

Map Üzerinde Gezinme

```
Map<String, Integer> notlar = new TreeMap<>();
notlar.put("Cem", 72);
notlar.put("Ayşe", 95);
notlar.put("Banu", 60);

for (String isim : notlar.keySet()) {
    System.out.print(isim + " ");
}
System.out.println();
for (Integer not : notlar.values()) {
    System.out.print(not + " ");
}
System.out.println();
for (Map.Entry<String, Integer> g : notlar.entrySet()) {
    System.out.println(g.getKey() + " → " + g.getValue());
}
```

```
Ayşe Banu Cem
95 60 72
Ayşe → 95
Banu → 60
Cem → 72
```

- `keySet()` : yalnız anahtarlar.
- `values()` : yalnız değerler.
- `entrySet()` : her çift bir `Map.Entry` ; `getKey()` ve `getValue()` ile okunur.
Hem anahtar hem değer gerekiyorsa en verimli yol budur.
- `Map` doğrudan `for-each` 'e verilemez: `Iterable` değildir.

Kelime Sayma: İzleme

```
► String metin = "elma armut elma kiraz armut elma";  
► Map<String, Integer> sayac = new TreeMap<>();  
3 for (String k : metin.split(" ")) {  
4     if (sayac.containsKey(k)) {  
5         sayac.put(k, sayac.get(k) + 1);  
6     } else {  
7         sayac.put(k, 1);  
8     }  
9 }  
10 System.out.println(sayac);
```

anahtar	değer
	boş

Metin ve boş bir `TreeMap` hazırlanır.
`split(" ")` metni kelimelere böler.

Adım 1 / 8

DEĞİŞKENLER

—

ÇIKTI

Kelime Sayma: İzleme

```
1 String metin = "elma armut elma kiraz armut elma";
2 Map<String, Integer> sayac = new TreeMap<>();
3 ▶ for (String k : metin.split(" ")) {
4   ▶ if (sayac.containsKey(k)) {
5     sayac.put(k, sayac.get(k) + 1);
6   } else {
7     ▶ sayac.put(k, 1);
8   }
9 }
10 System.out.println(sayac);
```

anahtar	değer
"elma"	1

"elma" ilk kez görüldü: `put(k, 1)`.

Adım 2 / 8

DEĞİŞKENLER

`k = "elma"`

ÇIKTI



Kelime Sayma: İzleme

```
1 String metin = "elma armut elma kiraz armut elma";
2 Map<String, Integer> sayac = new TreeMap<>();
3 ▶ for (String k : metin.split(" ")) {
4   ▶ if (sayac.containsKey(k)) {
5     sayac.put(k, sayac.get(k) + 1);
6   } else {
7     ▶ sayac.put(k, 1);
8   }
9 }
10 System.out.println(sayac);
```

anahtar	değer
"armut"	1
"elma"	1

"armut" ilk kez görüldü: `put(k, 1)`.

Adım 3 / 8

DEĞİŞKENLER

`k = "armut"`

ÇIKTI



Kelime Sayma: İzleme

```
1 String metin = "elma armut elma kiraz armut elma";
2 Map<String, Integer> sayac = new TreeMap<>();
3 ▶ for (String k : metin.split(" ")) {
4 ▶     if (sayac.containsKey(k)) {
5 ▶         sayac.put(k, sayac.get(k) + 1);
6     } else {
7         sayac.put(k, 1);
8     }
9 }
10 System.out.println(sayac);
```

anahtar	değer
"armut"	1
"elma"	2

"elma" zaten var: eski değer 1 alınır, 2 yazılır.

Adım 4 / 8

DEĞİŞKENLER

k = "elma"

ÇIKTI



Kelime Sayma: İzleme

```
1 String metin = "elma armut elma kiraz armut elma";
2 Map<String, Integer> sayac = new TreeMap<>();
3 ▶ for (String k : metin.split(" ")) {
4   ▶ if (sayac.containsKey(k)) {
5     sayac.put(k, sayac.get(k) + 1);
6   } else {
7     ▶ sayac.put(k, 1);
8   }
9 }
10 System.out.println(sayac);
```

anahtar	değer
"armut"	1
"elma"	2
"kiraz"	1

"kiraz" ilk kez görüldü: `put(k, 1)`.

Adım 5 / 8

DEĞİŞKENLER

`k = "kiraz"`

ÇIKTI



Kelime Sayma: İzleme

```
1 String metin = "elma armut elma kiraz armut elma";
2 Map<String, Integer> sayac = new TreeMap<>();
3 ▶ for (String k : metin.split(" ")) {
4 ▶     if (sayac.containsKey(k)) {
5 ▶         sayac.put(k, sayac.get(k) + 1);
6     } else {
7         sayac.put(k, 1);
8     }
9 }
10 System.out.println(sayac);
```

anahtar	değer
"armut"	2
"elma"	2
"kiraz"	1

"armut" zaten var: eski değer 1 alınır, 2 yazılır.

Adım 6 / 8

DEĞİŞKENLER

k = "armut"

ÇIKTI

Kelime Sayma: İzleme

```
1 String metin = "elma armut elma kiraz armut elma";
2 Map<String, Integer> sayac = new TreeMap<>();
3 ▶ for (String k : metin.split(" ")) {
4   ▶   if (sayac.containsKey(k)) {
5     ▶     sayac.put(k, sayac.get(k) + 1);
6   } else {
7     sayac.put(k, 1);
8   }
9 }
10 System.out.println(sayac);
```

anahtar	değer
"armut"	2
"elma"	3
"kiraz"	1

"elma" zaten var: eski değer 2 alınır, 3 yazılır.

Adım 7 / 8

DEĞİŞKENLER

k = "elma"

ÇIKTI



Kelime Sayma: İzleme

```
1 String metin = "elma armut elma kiraz armut elma";
2 Map<String, Integer> sayac = new TreeMap<>();
3 for (String k : metin.split(" ")) {
4     if (sayac.containsKey(k)) {
5         sayac.put(k, sayac.get(k) + 1);
6     } else {
7         sayac.put(k, 1);
8     }
9 }
▶ System.out.println(sayac);
```

anahtar	değer
"armut"	2
"elma"	3
"kiraz"	1

Döngü bitti. `TreeMap` anahtarları alfabetik sırayla yazdırır.

Adım 8 / 8

DEĞİŞKENLER

`k = "elma"`

ÇIKTI

```
{armut=2, elma=3, kiraz=1}
```

LinkedHashMap Sınıfı

- Tekil anahtar-değer çiftlerini içeren bir eşleme; `HashMap` 'ten türer.
- Her öge bir anahtar (key) ve bir değer (value) çiftidir; anahtarlar tekildir.
- Çiftleri bağlı listeyle birbirine bağlar: **eklenme sırası korunur.**
- Performans ve sıra birlikte önemliyse tercih edilir; sıra önemli değilse `HashMap` yeterlidir.

```
Map<String, String> rehber = new LinkedHashMap<>();
rehber.put("Zeynep", "555-1111");
rehber.put("Ahmet", "555-1234");
rehber.put("Mert", "555-2222");
Map<String, String> h = new HashMap<>(rehber);
Map<String, String> t = new TreeMap<>(rehber);
System.out.println("LinkedHashMap: " + rehber);
System.out.println("HashMap:      " + h);
System.out.println("TreeMap:      " + t);
```

```
LinkedHashMap: {Zeynep=555-1111, Ahmet=555-1234, Mert=555-2222}
HashMap:       {Ahmet=555-1234, Mert=555-2222, Zeynep=555-1111}
TreeMap:       {Ahmet=555-1234, Mert=555-2222, Zeynep=555-1111}
```

Temel metotlar `Map` ile aynıdır: `put`, `remove`, `get`, `containsKey`, `size`.

Hashtable Sınıfı

- Anahtar-değer çiftlerini saklayan, **senkronize** (thread-safe) eski bir eşleme sınıfıdır (Java 1.0).
- `HashMap` 'ten farkı: `HashMap` senkronize **değildir**; `Hashtable` 'ın her metodu kilitlidir.
- Anahtarlar da değerler de `null` olamaz.
- Kilitleme yüzünden modern alternatiflerden yavaştır; tek iş parçacığında `HashMap` , çok iş parçacığında `ConcurrentHashMap` tercih edilir.
- Metotlar: `put` , `remove` , `get` , `containsKey` , `size` .

```
Hashtable<String, String> telefonRehberi = new Hashtable<>();  
telefonRehberi.put("Ahmet", "555-1234");  
System.out.println(telefonRehberi.get("Ahmet"));  
System.out.println(telefonRehberi.containsKey("Ayşe"));  
// telefonRehberi.put(null, "x"); → NullPointerException
```

```
555-1234  
false
```

```
Exception in thread "main"  
java.lang.NullPointerException:  
Cannot invoke "Object.hashCode()" because "key" is null
```

SortedMap ve TreeMap

- `SortedMap` : anahtarları **sıralı** saklayan eşleme arayüzü; her anahtar bir kez bulunur.
- Alt harita ve üst harita gibi işlemler sunar.
- `TreeMap` : `AbstractMap` 'ten türetilir, `NavigableMap` arayüzünü gerçekleştirir.
- Anahtarlar doğal sıraya ya da verilen `Comparator` 'a göre dizilir.
- İçte **kırmızı-siyah ağaç** (red-black tree):
`get` / `put` $O(\log n)$.

Metot	Görevi
<code>firstKey()</code> / <code>lastKey()</code>	en küçük / en büyük anahtar
<code>headMap(to)</code>	<code>to</code> 'dan küçük anahtarlar (hariç)
<code>tailMap(from)</code>	<code>from</code> ve büyükleri (dahil)
<code>subMap(from, to)</code>	<code>from</code> dahil, <code>to</code> hariç
<code>comparator()</code>	sıralayan <code>Comparator</code> (doğal sırada <code>null</code>)
<code>put</code> , <code>get</code> , <code>remove</code>	<code>Map</code> ile aynı

EŞLEMELELER

TreeMap Örneği

```
TreeMap<String, Double> urunFiyatlari = new TreeMap<>();
urunFiyatlari.put("Telefon", 2000.0);
urunFiyatlari.put("Kalem", 15.0);
urunFiyatlari.put("Defter", 40.0);
urunFiyatlari.put("Silgi", 5.0);
System.out.println(urunFiyatlari);
System.out.println(urunFiyatlari.firstKey());
System.out.println(urunFiyatlari.lastKey());
System.out.println(urunFiyatlari.headMap("Kalem"));
System.out.println(urunFiyatlari.tailMap("Kalem"));
System.out.println(urunFiyatlari.subMap("Defter", "Silgi"));
```

ÇIKTI

```
{Defter=40.0, Kalem=15.0, Silgi=5.0, Telefon=2000.0}
Defter
Telefon
{Defter=40.0}
{Kalem=15.0, Silgi=5.0, Telefon=2000.0}
{Defter=40.0, Kalem=15.0}
```

EŞLEMELER

Map Sınıflarını Karşılaştırma

	HashMap	LinkedHashMap	TreeMap	Hashtable
Gezinme sırası	garanti yok	eklenme sırası	anahtar sıralı	garanti yok
<code>get</code> / <code>put</code>	O(1) ortalama	O(1) ortalama	O(log n)	O(1) ortalama, kilitli
<code>null</code> anahtar	bir tane	bir tane	hayır	hayır
<code>null</code> değer	evet	evet	evet	hayır
Senkronize	hayır	hayır	hayır	evet (eski)
Ne zaman?	varsayılan seçim	sıra korunmalı	sıralı anahtar, aralık sorgusu	eski kod

Aynı mantık kümelerde de geçerlidir: `HashSet`, `LinkedHashSet` ve `TreeSet` içte sırasıyla `HashMap`, `LinkedHashMap` ve `TreeMap` kullanır.

Iterator

- **Yineleyici** (iterator): koleksiyonda öğeleri **sırayla** gezmek için kullanılan nesne.
- Koleksiyonun iç yapısını bilmeden (dizi, ağaç, hash tablosu...) her öğeye erişmeyi sağlar.
- Gezerken güvenli **silme** yapılabilen tek yoldur.
- **for-each** döngüsü arka planda bir **Iterator** kullanır.
- Listeler için **ListIterator** ayrıca geriye gitmeyi, **add** ve **set** işlemlerini destekler.

Metot	Görevi
<code>hasNext()</code>	bir sonraki öğe var mı?
<code>next()</code>	sonraki öğeyi döndürür, imleci ilerletir
<code>remove()</code>	son <code>next()</code> ile alınan öğeyi koleksiyondan çıkarır

Genellikle `while (it.hasNext())` döngüsüyle (veya `for` ile) kullanılır.



Iterator: İzleme

```
▶ ArrayList<Integer> liste = new ArrayList<>();
▶ liste.add(3);
▶ liste.add(8);
▶ liste.add(5);

5 Iterator<Integer> iterator = liste.iterator();
6 while (iterator.hasNext()) {
7     Integer sayi = iterator.next();
8     // Eleman üzerinde işlemleri gerçekleştir.
9     System.out.println(sayi);
10 }
```



Liste hazırlanır: [3, 8, 5] .

Adım 1 / 9

DEĞİŞKENLER

—

ÇIKTI

Iterator: İzleme

```
1 ArrayList<Integer> liste = new ArrayList<>();
2 liste.add(3);
3 liste.add(8);
4 liste.add(5);
5 ▶ Iterator<Integer> iterator = liste.iterator();
6 while (iterator.hasNext()) {
7     Integer sayi = iterator.next();
8     // Eleman üzerinde işlemleri gerçekleştir.
9     System.out.println(sayi);
10 }
```

| imleç



`iterator()` : imleç ilk öğenin **önüne** konur.

Adım 2 / 9

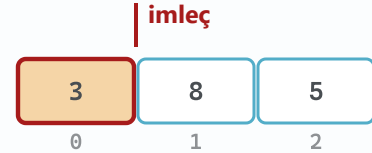
DEĞİŞKENLER

—

ÇIKTI

Iterator: İzleme

```
1 ArrayList<Integer> liste = new ArrayList<>();
2 liste.add(3);
3 liste.add(8);
4 liste.add(5);
5 Iterator<Integer> iterator = liste.iterator();
▶ while (iterator.hasNext()) {
▶     Integer sayi = iterator.next();
8     // Eleman üzerinde işlemleri gerçekleştir.
9     System.out.println(sayi);
10 }
```



`hasNext()` `true` ; `next()` 3'ü döndürür ve imleci ilerletir.

Adım 3 / 9

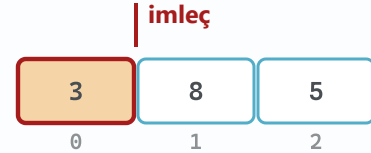
DEĞİŞKENLER

`sayi = 3`

ÇIKTI

Iterator: İzleme

```
1 ArrayList<Integer> liste = new ArrayList<>();
2 liste.add(3);
3 liste.add(8);
4 liste.add(5);
5 Iterator<Integer> iterator = liste.iterator();
6 while (iterator.hasNext()) {
7     Integer sayi = iterator.next();
8     // Eleman üzerinde işlemleri gerçekleştir.
9     ▶ System.out.println(sayi);
10 }
```



Öğeyazdırılır.

Adım 4 / 9

DEĞİŞKENLER

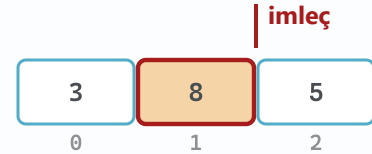
sayi = 3

ÇIKTI

3

Iterator: İzleme

```
1 ArrayList<Integer> liste = new ArrayList<>();
2 liste.add(3);
3 liste.add(8);
4 liste.add(5);
5 Iterator<Integer> iterator = liste.iterator();
▶ while (iterator.hasNext()) {
▶     Integer sayi = iterator.next();
8     // Eleman üzerinde işlemleri gerçekleştir.
9     System.out.println(sayi);
10 }
```



`hasNext()` `true` ; `next()` 8'i döndürür ve imleci ilerletir.

Adım 5 / 9

DEĞİŞKENLER

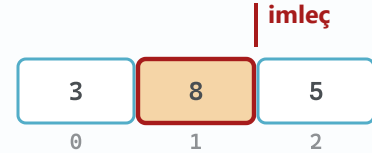
`sayi = 8`

ÇIKTI

3

Iterator: İzleme

```
1 ArrayList<Integer> liste = new ArrayList<>();
2 liste.add(3);
3 liste.add(8);
4 liste.add(5);
5 Iterator<Integer> iterator = liste.iterator();
6 while (iterator.hasNext()) {
7     Integer sayi = iterator.next();
8     // Eleman üzerinde işlemleri gerçekleştir.
9     ▶ System.out.println(sayi);
10 }
```



Öğeyazdırılır.

Adım 6 / 9

DEĞİŞKENLER

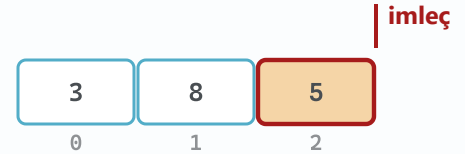
sayi = 8

ÇIKTI

3
8

Iterator: İzleme

```
1 ArrayList<Integer> liste = new ArrayList<>();
2 liste.add(3);
3 liste.add(8);
4 liste.add(5);
5 Iterator<Integer> iterator = liste.iterator();
▶ while (iterator.hasNext()) {
▶     Integer sayi = iterator.next();
8     // Eleman üzerinde işlemleri gerçekleştir.
9     System.out.println(sayi);
10 }
```



`hasNext()` `true` ; `next()` 5'i
döndürür ve imleci ilerletir.

Adım 7 / 9

DEĞİŞKENLER

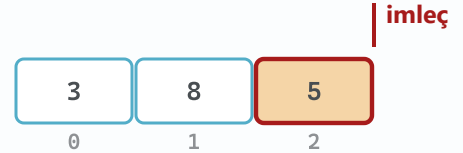
`sayi = 5`

ÇIKTI

3
8

Iterator: İzleme

```
1 ArrayList<Integer> liste = new ArrayList<>();
2 liste.add(3);
3 liste.add(8);
4 liste.add(5);
5 Iterator<Integer> iterator = liste.iterator();
6 while (iterator.hasNext()) {
7     Integer sayi = iterator.next();
8     // Eleman üzerinde işlemleri gerçekleştir.
9     ▶ System.out.println(sayi);
10 }
```



Öğeyazdırılır.

Adım 8 / 9

DEĞİŞKENLER

sayi = 5

ÇIKTI

3
8
5

Iterator: İzleme

```
1 ArrayList<Integer> liste = new ArrayList<>();
2 liste.add(3);
3 liste.add(8);
4 liste.add(5);
5 Iterator<Integer> iterator = liste.iterator();
▶ while (iterator.hasNext()) {
7     Integer sayi = iterator.next();
8     // Eleman üzerinde işlemleri gerçekleştir.
9     System.out.println(sayi);
10 }
```



İmleç sonda: `hasNext()` `false`, döngü biter.

Adım 9 / 9

DEĞİŞKENLER

`sayi = 5`

ÇIKTI

3
8
5

Gezinirken Silme

YANLIŞ: FOR-EACH İÇİNDE KOLEKSİYONDAN SİLMEK

```
List<Integer> sayilar = new ArrayList<>(Arrays.asList(4, 7, 10, 13, 16));
for (Integer s : sayilar) {
    if (s % 2 == 0) {
        sayilar.remove(s);
    }
}
```

Exception in thread "main" java.util.ConcurrentModificationException

DOĞRU: ITERATOR.REMOVE()

```
List<Integer> sayilar = new ArrayList<>();
Collections.addAll(sayilar, 4, 7, 10, 13, 16);
Iterator<Integer> it = sayilar.iterator();
while (it.hasNext()) {
    if (it.next() % 2 == 0) {
        it.remove(); // güvenli silme
    }
}
System.out.println(sayilar);
```

KISA YOL (JAVA 8+)

```
List<Integer> s2 = new ArrayList<>();
Collections.addAll(s2, 4, 7, 10, 13, 16);
s2.removeIf(s -> s % 2 == 0);
System.out.println(s2);
```

[7, 13]

İki yolun çıktısı da aynı: yalnız tek sayılar kalır.

for-each İçinde Silmek: İstisna Nereden Gelir?

```
► List<Integer> s = new ArrayList<>();  
► Collections.addAll(s, 6, 9, 12);  
3 for (Integer x : s) {           // gizli Iterator  
4     if (x % 2 == 0) {  
5         s.remove(x);  
6     }  
7 }  
8 System.out.println(s);
```

s (iç dizi)



modCount = 0

Liste 3 ögesi; **modCount** (yapısal değişiklik sayacı) 0.

Adım 1 / 8

DEĞİŞKENLER

s = [6, 9, 12]

modCount = 0

ÇIKTI

for-each İçinde Silmek: İstisna Nereden Gelir?

```
1 List<Integer> s = new ArrayList<>();
2 Collections.addAll(s, 6, 9, 12);
3 ► for (Integer x : s) {           // gizli Iterator
4     if (x % 2 == 0) {
5         s.remove(x);
6     }
7 }
8 System.out.println(s);
```

s (iç dizi)



size = 3

cursor = 0

modCount = 0	=
expected = 0	

for-each gizlice `s.iterator()` çağırır: cursor = 0 ve yineleyici o anki modCount'u saklar: expectedModCount = 0.

Adım 2 / 8

DEĞİŞKENLER

s = [6, 9, 12]

modCount = 0

ÇIKTI



for-each İçinde Silmek: İstisna Nereden Gelir?

```
1 List<Integer> s = new ArrayList<>();
2 Collections.addAll(s, 6, 9, 12);
3 ► for (Integer x : s) {           // gizli Iterator
4     if (x % 2 == 0) {
5         s.remove(x);
6     }
7 }
8 System.out.println(s);
```

s (iç dizi)



size = 3

cursor = 0

modCount = 0	=
expected = 0	

`hasNext()` : cursor (0) $\neq$ size (3) $\rightarrow$ true.

Adım 3 / 8

DEĞİŞKENLER

`s = [6, 9, 12]``modCount = 0``hasNext = true`

ÇIKTI



for-each İçinde Silmek: İstisna Nereden Gelir?

```
1 List<Integer> s = new ArrayList<>();
2 Collections.addAll(s, 6, 9, 12);
3 ► for (Integer x : s) {           // gizli Iterator
4     if (x % 2 == 0) {
5         s.remove(x);
6     }
7 }
8 System.out.println(s);
```

s (iç dizi)



modCount = 0
expected = 0 =

next() : önce sayaçlar karşılaştırılır (0 = 0); x = 6, cursor = 1.

Adım 4 / 8

DEĞİŞKENLER

s = [6, 9, 12]

modCount = 0

x = 6

ÇIKTI

for-each İçinde Silmek: İstisna Nereden Gelir?

```
1 List<Integer> s = new ArrayList<>();
2 Collections.addAll(s, 6, 9, 12);
3 for (Integer x : s) {           // gizli Iterator
4     if (x % 2 == 0) {
5         s.remove(x);
6     }
7 }
8 System.out.println(s);
```

s (iç dizi)



modCount = 0
expected = 0 =

6 çift.

Adım 5 / 8

DEĞİŞKENLER

s = [6, 9, 12] modCount = 0 x = 6

ÇIKTI

for-each İçinde Silmek: İstisna Nereden Gelir?

```
1 List<Integer> s = new ArrayList<>();
2 Collections.addAll(s, 6, 9, 12);
3 for (Integer x : s) {           // gizli Iterator
4     if (x % 2 == 0) {
5         s.remove(x);
6     }
7 }
8 System.out.println(s);
```

s (iç dizi)



modCount = 1
expected = 0

≠

`s.remove(x)` listenin kendi metodu: 9 ve 12 sola kayar, size 2, **modCount 1**. Yineleyicinin haberi yok.

Adım 6 / 8

DEĞİŞKENLER

s = [9, 12]

modCount = 1

x = 6

ÇIKTI

for-each İçinde Silmek: İstisna Nereden Gelir?

```
1 List<Integer> s = new ArrayList<>();
2 Collections.addAll(s, 6, 9, 12);
3
4 ▶ for (Integer x : s) { // gizli Iterator
5     if (x % 2 == 0) {
6         s.remove(x);
7     }
8 }
9 System.out.println(s);
```

s (iç dizi)



modCount = 1
expected = 0

≠

`hasNext()` : cursor (1) ≠ size (2) → true; döngü sürer.

Adım 7 / 8

DEĞİŞKENLER

`s = [9, 12]`

`modCount = 1`

`hasNext = true`

ÇIKTI

[9, 12]

for-each İçinde Silmek: İstisna Nereden Gelir?

```
1 List<Integer> s = new ArrayList<>();
2 Collections.addAll(s, 6, 9, 12);
3
4 ▶ for (Integer x : s) {           // gizli Iterator
5     if (x % 2 == 0) {
6         s.remove(x);
7     }
8 }
9 System.out.println(s);
```

s (iç dizi)



modCount = 1
expected = 0 ≠

ConcurrentModificationException

next() : modCount (1) ≠ expectedModCount (0) →
ConcurrentModificationException . Liste {6, 9}
olsaydı hasNext false dönerdi; istisna oluşmaz, 9 hiç
denetlenmezdi.

Adım 8 / 8

DEĞİŞKENLER

s = [9, 12]

modCount = 1

ÇIKTI

```
Exception in thread "main"
java.util.ConcurrentModificationException
```

Collections Sınıfı ve Algoritmalar

- `Collection` bir **arayüzdür**; `Collections` (sonunda **s**) ise koleksiyonlar üzerinde çalışan hazır `static` metotları toplayan bir **yardımcı sınıftır**.
- Bu hazır algoritmalar hızlı, etkili ve test edilmiştir; kendi döngünüzü yazmak yerine bunları kullanın.
- Doğru algoritmayı seçmek performansı önemli ölçüde etkiler.

Sıralama

`sort` ,
`reverse` ,
`reverseOrder`

Karıştırma

`shuffle`

Arama

`binarySearch` ,
`max` , `min` ,
`frequency`

Kompozisyon

`copy` , `fill` ,
`swap` ,
`reverse`

Temel Sıralama Algoritmaları

Algoritma	Fikir	Ortalama	En kötü	Not
Kabarcık (Bubble Sort)	komşu iki öğeyi karşılaştır, gerekirse yer değiştir	$O(n^2)$	$O(n^2)$	basit; büyük verilerde etkisiz
Seçmeli (Selection Sort)	en küçüğü bul, başa yerleştir; kalanla tekrarla	$O(n^2)$	$O(n^2)$	büyük verilerde yavaş
Ekleme (Insertion Sort)	sıradaki öğeyi sıralı kısımda uygun yere ekle	$O(n^2)$	$O(n^2)$	küçük veya neredeyse sıralı veride etkili
Birleştirme (Merge Sort)	ikiye böl, her yarıyı özyinelemeli sırala, birleştir	$O(n \log n)$	$O(n \log n)$	kararlı; büyük verilerde etkili
Hızlı (Quick Sort)	pivot seç, küçükleri sola büyükleri sağa ayır (böl ve yönet)	$O(n \log n)$	$O(n^2)$	ortalamada çok hızlı

`Collections.sort` nesneler için **TimSort** kullanır: birleştirme ve ekleme sıralamasının karması, kararlı, $O(n \log n)$. (`Arrays.sort` ilkel tiplerde Dual-Pivot Quicksort kullanır.) **Kararlı** (stable): eşit öğelerin göreceli sırası korunur.

Sıralama Metotları

```
List<String> isimler = new ArrayList<>(  
    Arrays.asList("Cem", "Ayşe", "Banu", "Ali"));  
Collections.sort(isimler);  
System.out.println(isimler);  
Collections.sort(isimler, Collections.reverseOrder());  
System.out.println(isimler);  
Collections.reverse(isimler);  
System.out.println(isimler);
```

```
[Ali, Ayşe, Banu, Cem]  
[Cem, Banu, Ayşe, Ali]  
[Ali, Ayşe, Banu, Cem]
```

Metot	Görevi
<code>sort(list)</code>	doğal sıraya göre; öğeler <code>Comparable</code> olmalı
<code>sort(list, c)</code>	verilen karşılaştırıcıya (<code>Comparator</code>) göre
<code>reverse(list)</code>	listeyi ters sıraya çevirir

`String` , `Integer` , `Double` gibi sınıflar zaten `Comparable` 'dır. Java 8+:
`liste.sort(null)` de doğal sıraya göre sıralar.

Tam imzalar: `sort(List<T> list)` , `sort(List<T> list, Comparator<? super T> c)` ,
`reverse(List<?> list)` .

Comparable: Doğal Sıra

```
class Ogrenci implements Comparable<Ogrenci> {
    String isim;
    int puan;

    Ogrenci(String isim, int puan) {
        this.isim = isim;
        this.puan = puan;
    }

    // puana göre artan sıra
    @Override
    public int compareTo(Ogrenci diger) {
        return Integer.compare(puan, diger.puan);
    }

    @Override
    public String toString() {
        return isim + ":" + puan;
    }
}
```

```
List<Ogrenci> sinif = new ArrayList<>();
sinif.add(new Ogrenci("Cem", 72));
sinif.add(new Ogrenci("Ayşe", 95));
sinif.add(new Ogrenci("Banu", 60));
```

```
Collections.sort(sinif);
System.out.println(sinif);
```

[Banu:60, Cem:72, Ayşe:95]

- Sınıf `Comparable<Ogrenci>` arayüzünü gerçekleştirir ve `compareTo` 'yu yazar.
- Sonuç: negatif → `this` önce; 0 → eşit; pozitif → `diger` önce.
- `Integer.compare` çıkarma yapmaktan (`puan - diger.puan`) güvenlidir: taşıma olmaz.

Comparator: Başka Bir Sıra

```
Collections.sort(sinif, new Comparator<Ogrenci>() {  
    @Override  
    public int compare(Ogrenci a, Ogrenci b) {  
        return a.isim.compareTo(b.isim);    // isme göre  
    }  
});  
System.out.println(sinif);  
  
// Java 8+: aynı işin kısa yazımı  
sinif.sort(Comparator.comparing(o -> o.puan));  
System.out.println(sinif);
```

```
[Ayşe:95, Banu:60, Cem:72]  
[Banu:60, Cem:72, Ayşe:95]
```

- **Comparator** : sınıfın dışında yazılan ayrı bir sıralama ölçütü; bir sınıf için birden çok olabilir.
- Klasik yazım: anonim iç sınıf (anonymous class) ile **compare(a, b)** .
- Java 8+:
Comparator.comparing(...) ve lambda ile tek satır; ters sıra için **.reversed()** .
- Doğal sıra (**compareTo**)
değişmez; yalnızca o sıralama için başka ölçüt kullanılır.

Karıştırma

- Koleksiyon öğelerini **rastgele** bir sıraya sokmak için kullanılır.
- `Collections.shuffle(liste)` **Fisher-Yates** (Knuth) algoritmasını kullanır: listenin sonundan başlayarak her öğeyi rastgele seçilen bir öğeyle yer değiştirir.
- Hızlı ve etkilidir: $O(n)$. Her olası sıralama **eşit olasılıkla** ortaya çıkar (düzgün dağılım).
- Kullanım: kart dağıtma, kura çekme, rastgele soru sırası.

```
List<String> ogrenciler = new ArrayList<>(  
    Arrays.asList("Ali", "Banu", "Cem", "Deniz"));  
Collections.shuffle(ogrenciler);  
System.out.println(ogrenciler);
```

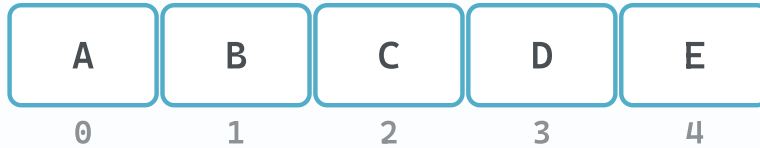
BİR ÇALIŞTIRMANIN ÇIKTISI

```
[Cem, Ali, Deniz, Banu]
```

Çıktı her çalıştırmada farklıdır. Aynı karışım için (test) sabit tohumlu `new Random(42)` ikinci argüman olarak verilir.

Fisher-Yates Adım Adım

Kural: i sondan 1'e kadar iner; her adımda 0 ile i arasında rastgele bir j seçilir ve $a[i]$ ile $a[j]$ yer değiştirir. Aşağıdaki j değerleri bir örnek çekiliştir.

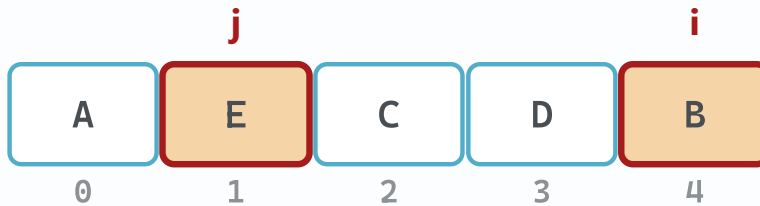


Başlangıç: $[A, B, C, D, E]$.

Adım 1 / 5

Fisher–Yates Adım Adım

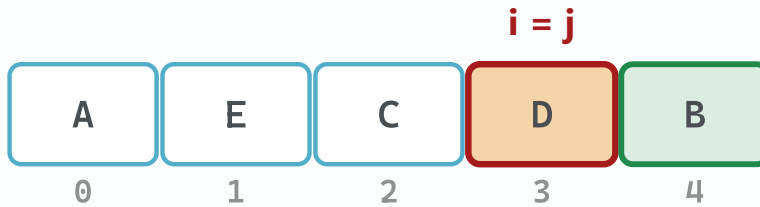
Kural: i sondan 1'e kadar iner; her adımda 0 ile i arasında rastgele bir j seçilir ve $a[i]$ ile $a[j]$ yer değiştirir. Aşağıdaki j değerleri bir örnek çekiliştir.



$i = 4$, rastgele $j = 1$: E ile B yer değiştirir. 4. konum artık kesinleşti.

Fisher-Yates Adım Adım

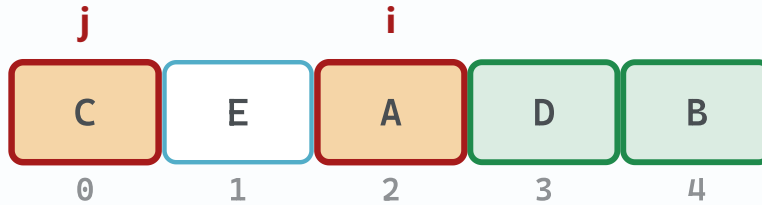
Kural: i sondan 1'e kadar iner; her adımda 0 ile i arasında rastgele bir j seçilir ve $a[i]$ ile $a[j]$ yer değiştirir. Aşağıdaki j değerleri bir örnek çekiliştir.



$i = 3$, $j = 3$: öge kendisiyle değişir (bu da olasıdır). Yeşil: kesinleşen konumlar.

Fisher-Yates Adım Adım

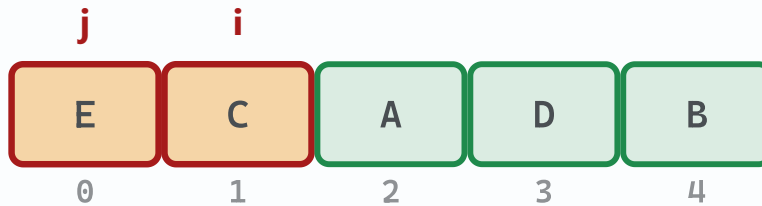
Kural: i sondan 1'e kadar iner; her adımda 0 ile i arasında rastgele bir j seçilir ve $a[i]$ ile $a[j]$ yer değiştirir. Aşağıdaki j değerleri bir örnek çekiliştir.



$i = 2$, $j = 0$: A ile C yer değiştirir.

Fisher–Yates Adım Adım

Kural: i sondan 1'e kadar iner; her adımda 0 ile i arasında rastgele bir j seçilir ve $a[i]$ ile $a[j]$ yer değiştirir. Aşağıdaki j değerleri bir örnek çekiliştir.



$i = 1$, $j = 0$: C ile E yer değiştirir. Sonuç: $[E, C, A, D, B]$; $n - 1$ değişim yeterli.

Loto Örneği

```
public class Loto {  
    // Loto çekilişi yapan metot  
    public static Integer[] lotoYap(int toplamSayi, int cekilecekSayi) {  
        ArrayList<Integer> sayilar = new ArrayList<Integer>();  
        for (int i = 0; i < toplamSayi; i++) {  
            sayilar.add(i + 1);          // 1 ... toplamSayi  
        }  
        Collections.shuffle(sayilar);    // karıştır  
        Integer[] cekilenSayilar = new Integer[cekilecekSayi];  
        for (int i = 0; i < cekilecekSayi; i++) {  
            cekilenSayilar[i] = sayilar.get(i);  
        }  
        return cekilenSayilar;  
    }  
  
    public static void main(String[] args) {  
        Integer[] sonuc = lotoYap(49, 6);    // 49'dan 6  
        for (Integer item : sonuc) {  
            System.out.print(item + " ");  
        }  
    }  
}
```

ÖRNEK ÇIKTI

46 18 22 11 20 10

1–49 listesi
karıştırılır, ilk 6 öğe
alınır: sayılar
tekrarsızdır. Çıktı
her çalıştırmada
farklıdır.

Arama Algoritmaları

Doğrusal arama (Linear Search)

Öğeleri baştan sona tek tek kontrol eder. Sıralı olma şartı yoktur; küçük koleksiyonlarda etkilidir. $O(n)$. Örnek:

`indexOf` , `contains` .

İkili arama (Binary Search)

Sıralı koleksiyonda ortadaki öğeye bakar, aranan küçükse sola, büyükse sağa gider; her adımda aralık yarıya iner. $O(\log n)$.

Koleksiyon sıralı olmalıdır.

`Collections.binarySearch()` sıralı bir listede ikili arama yapar. 1 milyon öğede doğrusal arama 1 milyona kadar, ikili arama en fazla 20 karşılaştırma yapar.

ALGORİTMALAR

İkili Arama Adım Adım

```
List<Integer> sayilar = Arrays.asList(1, 3, 5, 7, 9, 11);  
int indeks = Collections.binarySearch(sayilar, 7);  
System.out.println(indeks);  
System.out.println(Collections.binarySearch(sayilar, 6));  
System.out.println(sayilar.indexOf(7));    // doğrusal arama
```



7 aranıyor. `low = 0` , `high = 5` , `mid = 2` : $5 < 7 \rightarrow$
sağ yarıya: `low = 3` .

Adım 1 / 4

ÇIKTI

```
3  
-4  
3
```

Bulunamazsa dönüş

negatiftir:

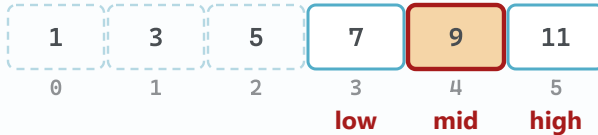
–(ekleme noktası) – 1. Liste
sıralı değilse sonuç

tanımsızdır.

ALGORİTMALAR

İkili Arama Adım Adım

```
List<Integer> sayilar = Arrays.asList(1, 3, 5, 7, 9, 11);  
int indeks = Collections.binarySearch(sayilar, 7);  
System.out.println(indeks);  
System.out.println(Collections.binarySearch(sayilar, 6));  
System.out.println(sayilar.indexOf(7));    // doğrusal arama
```



mid = 4 : 9 > 7 → sol yarıya: high = 3 .

Adım 2 / 4

ÇIKTI

3
-4
3

Bulunamazsa dönüş

negatiftir:

–(ekleme noktası) – 1. Liste
sıralı değilse sonuç

tanımsızdır.

ALGORİTMALAR

İkili Arama Adım Adım

```
List<Integer> sayilar = Arrays.asList(1, 3, 5, 7, 9, 11);  
int indeks = Collections.binarySearch(sayilar, 7);  
System.out.println(indeks);  
System.out.println(Collections.binarySearch(sayilar, 6));  
System.out.println(sayilar.indexOf(7));    // doğrusal arama
```



low=mid=high

mid = 3 : 7 bulundu → indis **3** döner. Yalnızca 3 karşılaştırma.

Adım 3 / 4

ÇIKTI

3
-4
3

Bulunamazsa dönüş

negatiftir:

–(ekleme noktası) – 1. Liste sıralı değilse sonuç

tanımsızdır.

İkili Arama Adım Adım

```
List<Integer> sayilar = Arrays.asList(1, 3, 5, 7, 9, 11);  
int indeks = Collections.binarySearch(sayilar, 7);  
System.out.println(indeks);  
System.out.println(Collections.binarySearch(sayilar, 6));  
System.out.println(sayilar.indexOf(7));    // doğrusal arama
```



6 aranırsa bulunamaz; 6'nın girmesi gereken yer 3'tür.

Dönüş: $-(3) - 1 = -4$.

Adım 4 / 4

ÇIKTI

3
-4
3

Bulunamazsa dönüş

negatiftir:

$-(\text{ekleme noktası}) - 1$. Liste
sıralı değilse sonuç

tanımsızdır.

Kompozisyon Algoritmaları

Metot	Görevi
<code>copy(hedef, kaynak)</code>	kaynağın öğelerini hedefin başına kopyalar; <code>hedef.size() ≥ kaynak.size()</code> olmalı
<code>fill(liste, d)</code>	tüm öğeleri <code>d</code> yapar
<code>reverse(liste)</code>	öğelerin sırasını tersine çevirir
<code>swap(liste, i, j)</code>	<code>i</code> ve <code>j</code> konumlarını değiştirir
<code>max</code> / <code>min</code>	en büyük / en küçük öğe
<code>frequency(c, o)</code>	<code>o</code> 'nun kaç kez geçtiği

```
List<Integer> n = Arrays.asList(4, 9, 2, 9, 5);
System.out.println(Collections.max(n));
System.out.println(Collections.min(n));
System.out.println(Collections.frequency(n, 9));
```

```
9
2
2
```

Hedef kısa ise `copy`

`IndexOutOfBoundsException: Source does not fit in dest` fırlatır.

Kompozisyon: İzleme

```
▶ List<String> isimler =  
▶     Arrays.asList("Ahmet", "Ayşe", "Cem");  
▶ List<String> hedef = new ArrayList<>(  
▶     Arrays.asList("X", "Y", "Z", "W"));  
5 Collections.copy(hedef, isimler);  
6 System.out.println(hedef);  
7 Collections.reverse(hedef);  
8 System.out.println(hedef);  
9 Collections.swap(hedef, 0, 2);  
10 System.out.println(hedef);  
11 Collections.fill(hedef, "Java");  
12 System.out.println(hedef);
```

isimler

Ahmet

Ayşe

Cem

hedef

X

Y

Z

W

0

1

2

3

`Arrays.asList` sabit boyutlu bir liste;
`hedef` ise 4 öğeli bir `ArrayList`.

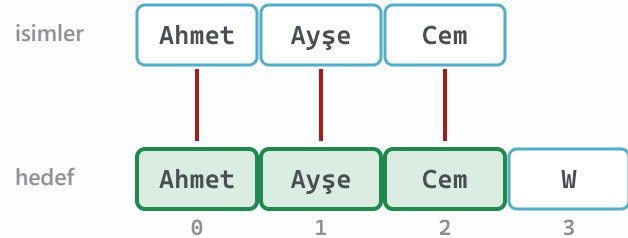
Adım 1 / 5

ÇIKTI



Kompozisyon: İzleme

```
1 List<String> isimler =  
2     Arrays.asList("Ahmet", "Ayşe", "Cem");  
3 List<String> hedef = new ArrayList<>(  
4     Arrays.asList("X", "Y", "Z", "W"));  
▶ Collections.copy(hedef, isimler);  
▶ System.out.println(hedef);  
7 Collections.reverse(hedef);  
8 System.out.println(hedef);  
9 Collections.swap(hedef, 0, 2);  
10 System.out.println(hedef);  
11 Collections.fill(hedef, "Java");  
12 System.out.println(hedef);
```



copy : 3 öğe hedefin 0–2 konumlarına yazılır; W yerinde kalır.

Adım 2 / 5

ÇIKTI

[Ahmet, Ayşe, Cem, W]

Kompozisyon: İzleme

```
1 List<String> isimler =  
2     Arrays.asList("Ahmet", "Ayşe", "Cem");  
3 List<String> hedef = new ArrayList<>(  
4     Arrays.asList("X", "Y", "Z", "W"));  
5 Collections.copy(hedef, isimler);  
6 System.out.println(hedef);  
7  
8 ▶ Collections.reverse(hedef);  
9 ▶ System.out.println(hedef);  
10  
11 Collections.swap(hedef, 0, 2);  
12 System.out.println(hedef);
```

isimler

Ahmet

Ayşe

Cem

hedef

W

Cem

Ayşe

Ahmet

0

1

2

3

reverse : sıra tersine döner.

Adım 3 / 5

ÇIKTI

[Ahmet, Ayşe, Cem, W]

[W, Cem, Ayşe, Ahmet]

Kompozisyon: İzleme

```
1 List<String> isimler =  
2     Arrays.asList("Ahmet", "Ayşe", "Cem");  
3 List<String> hedef = new ArrayList<>(  
4     Arrays.asList("X", "Y", "Z", "W"));  
5 Collections.copy(hedef, isimler);  
6 System.out.println(hedef);  
7 Collections.reverse(hedef);  
8 System.out.println(hedef);  
▶ Collections.swap(hedef, 0, 2);  
▶ System.out.println(hedef);  
11 Collections.fill(hedef, "Java");  
12 System.out.println(hedef);
```

isimler

Ahmet

Ayşe

Cem

hedef

Ayşe

Cem

W

Ahmet

0

1

2

3

`swap(hedef, 0, 2)` : 0 ve 2 konumları yer değiştirir.

Adım 4 / 5

ÇIKTI

```
[Ahmet, Ayşe, Cem, W]
```

```
[W, Cem, Ayşe, Ahmet]
```

```
[Ayşe, Cem, W, Ahmet]
```

Kompozisyon: İzleme

```
1 List<String> isimler =  
2     Arrays.asList("Ahmet", "Ayşe", "Cem");  
3 List<String> hedef = new ArrayList<>(  
4     Arrays.asList("X", "Y", "Z", "W"));  
5 Collections.copy(hedef, isimler);  
6 System.out.println(hedef);  
7 Collections.reverse(hedef);  
8 System.out.println(hedef);  
9 Collections.swap(hedef, 0, 2);  
10 System.out.println(hedef);  
▶ Collections.fill(hedef, "Java");  
▶ System.out.println(hedef);
```

isimler

Ahmet

Ayşe

Cem

hedef

Java

Java

Java

Java

0

1

2

3

fill : tüm öğeler **"Java"** olur. Boyut değişmez.

Adım 5 / 5

ÇIKTI

```
[Ahmet, Ayşe, Cem, W]  
[W, Cem, Ayşe, Ahmet]  
[Ayşe, Cem, W, Ahmet]  
[Java, Java, Java, Java]
```

SEÇİM

Hangi Koleksiyonu Seçmeli?

İhtiyaç	Seçim	Neden
Sıralı liste, indisle erişim	<code>ArrayList</code>	hızlı <code>get(i)</code> ; varsayılan liste
Başa/sona sık ekleme-çıkarma, kuyruk, yığın	<code>ArrayDeque</code>	iki uç $O(1)$; <code>Stack</code> ve <code>LinkedList</code> 'ten hızlı
En öncelikli öğeyi sürekli almak	<code>PriorityQueue</code>	<code>poll</code> en küçüğü (veya ölçüte göre en önceliklisini) verir
Tekrarsız öğeler, hızlı "var mı?"	<code>HashSet</code>	<code>contains</code> $O(1)$ ortalama
Tekrarsız + eklenme sırası	<code>LinkedHashSet</code>	sıra korunur
Tekrarsız + sıralı, aralık sorgusu	<code>TreeSet</code>	<code>first</code> , <code>headSet</code> , <code>subSet</code>
Anahtardan değere hızlı erişim	<code>HashMap</code>	<code>get</code> $O(1)$ ortalama; varsayılan eşleme
Anahtarlar sıralı	<code>TreeMap</code>	<code>firstKey</code> , <code>headMap</code> , <code>subMap</code>

Değişkeni arayüz tipinde tanımlayın (`List` , `Set` , `Map` , `Queue` , `Deque`): ihtiyaç değişirse yalnızca `new` satırı değişir.

SEÇİM

Değiştirilemez Koleksiyonlar

FABRİKA METOTLARI (JAVA 9+)

```
List<String> r = List.of("kırmızı", "yeşil");
Set<Integer> asal = Set.of(2, 3, 5);
Map<String, Integer> puan =
    Map.of("Ali", 90, "Banu", 85);
System.out.println(r);
System.out.println(puan.get("Banu"));
r.add("mavi");
```

```
[kırmızı, yeşil]
85
```

```
Exception in thread "main"
java.lang.UnsupportedOperationException
```

`add` , `remove` , `put` istisna fırlatır.
`null` öge kabul edilmez; `Set.of(1, 1)` → `IllegalArgumentException` .

SALT OKUNUR GÖRÜNÜM (JAVA 8)

```
List<String> liste = new ArrayList<>();
liste.add("a");
List<String> salt =
    Collections.unmodifiableList(liste);
liste.add("b");
System.out.println(salt);
salt.add("c");
```

```
[a, b]
```

```
Exception in thread "main"
java.lang.UnsupportedOperationException
```

`salt` bir **görünümdür**: kendisi değiştirilemez, ama alttaki `liste` değişirse o da değişir. Bağımsız kopya: `List.copyOf(liste)` (Java 10+). Sabit listeler ve hash anahtarları için uygundur.

ÖZET

Özet

Sınıf	Arayüz	Sıra	Tekrar	Temel yapı
<code>ArrayList</code>	List	eklenme	evet	dinamik dizi
<code>LinkedList</code>	List, Deque	eklenme	evet	çift yönlü bağlı liste
<code>Vector</code> / <code>Stack</code>	List	eklenme / LIFO	evet	senkronize dizi (eski)
<code>ArrayDeque</code>	Deque	FIFO veya LIFO	evet	dairesel dizi
<code>PriorityQueue</code>	Queue	önceliğe göre çıkış	evet	ikili yığın
<code>HashSet</code>	Set	garanti yok	hayır	hash tablosu
<code>LinkedHashSet</code>	Set	eklenme	hayır	hash + bağlı liste
<code>TreeSet</code>	NavigableSet	sıralı	hayır	kırmızı-siyah ağaç
<code>HashMap</code>	Map	garanti yok	anahtar hayır	hash tablosu
<code>LinkedHashMap</code>	Map	eklenme	anahtar hayır	hash + bağlı liste
<code>TreeMap</code>	NavigableMap	anahtar sıralı	anahtar hayır	kırmızı-siyah ağaç

Gezinme: `for-each` veya `Iterator` (silme için). Algoritmalar: `Collections.sort`, `shuffle`, `binarySearch`, `copy`, `fill`, `reverse`, `swap`.

ÖZET

İlgili Sayfalar

Koleksiyonların İç Yapısı

Simülatör: ArrayList
kapasitesi ve kaydırma,
HashMap kovaları, çakışma
zinciri ve yeniden
boyutlandırma.

Çalışma Soruları

Koleksiyon seçimi, çıktı
soruları ve ArrayList,
ArrayDeque, TreeMap
üzerinde adım adım
izlemeler.

Terim Sözlüğü

Liste, küme, eşleme, kuyruk,
yineleyici ve jenerik tip gibi
terimlerin tek Türkçe
karşılıkları.

İpucu: koleksiyonlarla çalışırken önce doğru arayüzü seçin, sonra uygulamayı. Jenerik tipler hataları derleme zamanında yakalar.

Sık Yapılan Hatalar (1/4)

İLKEL TİP İLE JENERİK

```
List<int> notlar = new ArrayList<>();
```

```
error: unexpected type
      List<int> notlar = new ArrayList<>();
      ^
required: reference
found:    int
```

Koleksiyonlar nesne tutar:

`List<Integer>` yazın. `int` değerler otomatik kutulanır.

YANLIŞ TİPTE ÖĞE EKLEMEK

```
List<String> isimler = new ArrayList<>();
isimler.add(42);
```

```
error: incompatible types: int cannot be
converted to String
      isimler.add(42);
                  ^
```

Jenerik tip derleme zamanında denetlenir.

Sayıyı dizgi olarak saklamak isterseniz

```
String.valueOf(42) .
```

Sık Yapılan Hatalar (2/4)

REMOVE(İNT) İLE REMOVE(OBJECT) KARIŞMASI

```
List<Integer> liste = new ArrayList<>();
Collections.addAll(liste, 10, 20, 30);
liste.remove(1); // indeks 1
System.out.println(liste);
liste.remove(Integer.valueOf(10)); // değer 10
System.out.println(liste);
```

[10, 30]
[30]

`List<Integer>` 'da `remove(1)` **indis** 1'i siler. Değeri silmek için `Integer.valueOf(...)` verin.

INTEGER'LARI == İLE KARŞILAŞTIRMAK

```
List<Integer> a = new ArrayList<>();
Collections.addAll(a, 1000, 1000);
System.out.println(a.get(0) == a.get(1));
System.out.println(a.get(0).equals(a.get(1)));
```

false
true

`==` referansları karşılaştırır (–128...127 arası önbellekten dolayı tesadüfen doğru çıkabilir). Değer için `equals` kullanın.

Sık Yapılan Hatalar (3/4)

ARRAYS.asList LİSTESİNE EKLEMEK

```
List<Integer> sabit = Arrays.asList(1, 2, 3);  
sabit.add(4);
```

```
Exception in thread "main"  
java.lang.UnsupportedOperationException
```

`Arrays.asList` **sabit boyutlu** liste döndürür (`set` olur, `add` / `remove` olmaz). Çözüm: `new ArrayList<>(Arrays.asList(...))` .

MAP'İ DOĞRUDAN FOR-EACH İLE GEZMEK

```
Map<String, Integer> m = new HashMap<>();  
for (String s : m) {  
    System.out.println(s);  
}
```

```
error: for-each not applicable to expression  
type
```

```
    for (String s : m) {  
                    ^
```

```
required: array or java.lang.Iterable  
found:    Map<String,Integer>
```

`Map` `Iterable` değildir: `keySet()` , `values()` veya `entrySet()` üzerinde gezin.

Sık Yapılan Hatalar (4/4)

VAR OLMAYAN İNDİS

```
List<String> l = new ArrayList<>();  
Collections.addAll(l, "a", "b", "c");  
System.out.println(l.get(3));
```

```
Exception in thread "main"  
java.lang.IndexOutOfBoundsException:  
Index 3 out of bounds for length 3
```

Geçerli indisler 0 ... `size() - 1` .
Boş kuyrukta `remove()` da
`NoSuchElementException` fırlatır;
`poll()` `null` döner.

TÜRKÇE ADLARI SIRALAMAK

```
List<String> adlar = new ArrayList<>();  
Collections.addAll(adlar,  
    "Zeynep", "Çağla", "Can", "Ömer");  
Collections.sort(adlar);  
System.out.println(adlar);  
Locale tr = Locale.forLanguageTag("tr-TR");  
Collections.sort(adlar, Collator.getInstance(tr));  
System.out.println(adlar);
```

```
[Can, Zeynep, Çağla, Ömer]  
[Can, Çağla, Ömer, Zeynep]
```

`String` karakter kodlarıyla
sıralanır: Ç, Ö, Ş, Ü, İ "Z"den sonra
gelir. Türkçe alfabe sırası için
`Collator` (java.text) kullanın.

Örnek Problemler (1)

```
public class Fibonacci {  
    public static int fibonacci(int sayi) {  
        List<Integer> liste = new ArrayList<>();  
        liste.add(0);  
        liste.add(1);  
        for (int i = 2; i < sayi + 1; i++) {  
            liste.add(liste.get(i - 2) + liste.get(i - 1));  
        }  
        return liste.get(sayi);  
    }  
    public static void main(String[] args) {  
        int sonuc = fibonacci(10);  
        System.out.println(sonuc);  
    }  
}
```

1. Program ne yazdırır?
2. `fibonacci(10)` dönerken listede kaç öge vardır? Son üç öge nedir?
3. `fibonacci(1)` çağrısında döngü kaç kez döner?
4. Bu soru için liste gerçekten gerekli mi?

► Cevapları göster

Örnek Problemler (2)

```
public class TekrarSiz {  
    static List<String> tekrarSiz(List<String> giris) {  
        Set<String> gorulen = new LinkedHashSet<>(giris);  
        return new ArrayList<>(gorulen);  
    }  
  
    public static void main(String[] args) {  
        List<String> l = Arrays.asList("b", "a", "b", "c", "a");  
        System.out.println(tekrarSiz(l));  
    }  
}
```

1. Program ne yazdırır?
2. `LinkedHashSet` yerine `HashSet` veya `TreeSet` kullanılsaydı ne değişirdi?

► Cevapları göster

Örnek Problemler (3)

```
static boolean dengeliMi(String ifade) {  
    Deque<Character> yigin = new ArrayDeque<>();  
    for (char c : ifade.toCharArray()) {  
        if (c == '(' || c == '[') {  
            yigin.push(c);  
        } else if (c == ')' || c == ']') {  
            if (yigin.isEmpty()) return false;  
            char ac = yigin.pop();  
            if (c == ')' && ac != '(') return false;  
            if (c == ']' && ac != '[') return false;  
        }  
    }  
    return yigin.isEmpty();  
}
```

Parantez dengesi: her açılan parantez doğru türde ve doğru sırada kapanmalı.

1. `dengeliMi("a[b]c")` , `dengeliMi("a[b]c")` ve `dengeliMi("(")` ne döndürür?
2. Neden yığın kullanıldı?

► Cevapları göster

Örnek Problemler (4)

```
Deque<Integer> d = new ArrayDeque<>();  
d.offerLast(1);  
d.offerFirst(2);  
d.offerLast(3);  
d.push(4);  
System.out.println(d);  
System.out.println(d.pollLast() + d.pop());  
System.out.println(d);
```

```
Map<String, Integer> m = new TreeMap<>();  
m.put("b", 1);  
m.put("a", 2);  
m.put("b", 3);  
Set<Integer> s = new TreeSet<>(m.values());  
s.add(2);  
System.out.println(m + " " + s + " " + m.size());
```

Her iki parçanın çıktısını tahmin edin. (İpucu:

`push` = `offerFirst` , `pop` = `removeFirst` .)

► Cevapları göster

Alıştırmalar (1)

1. Hangi koleksiyon? (a) numaradan öğrenciye hızlı erişim, (b) farklı kelimeleri alfabetik listelemek, (c) yazıcı işleri, (d) tarayıcının “geri” düğmesi, (e) acil serviste aciliyete göre çağırma.
2. Bir `String[]` dizisinde en sık geçen kelimeyi döndüren `enSikKelime` metodunu bir `HashMap` ile yazın.

► Cevapları göster

Alıştırmalar (2)

3. Loto çekilişini `shuffle` kullanmadan, `Random` ve bir `TreeSet` ile yazın: 1–49 arasından 6 farklı sayı, sıralı yazdırılsın.

► Cevapları göster

Alıştırmalar (3)

4. Her öğrencinin not listesini tutan bir `Map<String, List<Integer>>` kurun; Ali: 80, 90 ve Banu: 70 için ortalamaları yazdırın.

► Cevapları göster

Kaynaklar

- Oracle, *The Java Tutorials*: Collections (Interfaces, Implementations, Algorithms) ve Generics.
- Oracle, *Java SE 21 API Documentation*: `java.util` paketi (`Collection` , `List` , `Map` , `Collections` ...).
- J. Gosling ve diğ., *The Java Language Specification*, Java SE 21 Edition (4.5 Parameterized Types, 5.1.7 Boxing Conversion, 14.14.2 Enhanced for).
- P. Deitel ve H. Deitel, *Java: How to Program*, 11. baskı, Pearson (Bölüm 16: Generic Collections).
- K. Sierra ve B. Bates, *Head First Java*, 3. baskı, O'Reilly (Bölüm 11: Data Structures).
- C. S. Horstmann, *Core Java, Volume I: Fundamentals*, 12. baskı, Pearson (Bölüm 9: Collections).
- M. Naftalin ve P. Wadler, *Java Generics and Collections*, O'Reilly.
- J. Bloch, *Effective Java*, 3. baskı, Addison-Wesley (Madde 11: equals ile birlikte hashCode; Madde 64: nesnelere arayüzleriyle başvurun).

Son

Bölüm 5: Koleksiyonlar

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