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 | Akış API'si

Sercan KÜLCÜ · Giresun Üniversitesi · Bilgisayar Mühendisliği

© Sercan Külcü. Tüm hakları saklıdır. · Sunumun etkileşimli HTML sürümü: https://sercankulcu.github.io/files/java/slides/Bolum_05_Koleksiyonlar.html

GİRİŞ

Öğrenme Çıktıları

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

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

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

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

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

Gezinme, algoritmalar, akışlar
`Iterator` ile güvenle gezinmek; `Collections` ile sıralamak, karıştırmak, aramak; `java.util.stream` ile süzme, dönüştürme ve toplama hatları kurmak.

Neden Koleksiyonlar?

- **Koleksiyon (collection)**: birden çok öğeyi 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.

KOLEKSİYON ÇERÇEVESİ

Koleksiyon Türleri

List (liste)

Sıralı, indisli öğeler; tekrar olabilir. `ArrayList`, `LinkedList`



Set (küme)

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



Map (eşleme)

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



Queue (kuyruk)

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



KOLEKSİYON ÇERÇEVESİ

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.

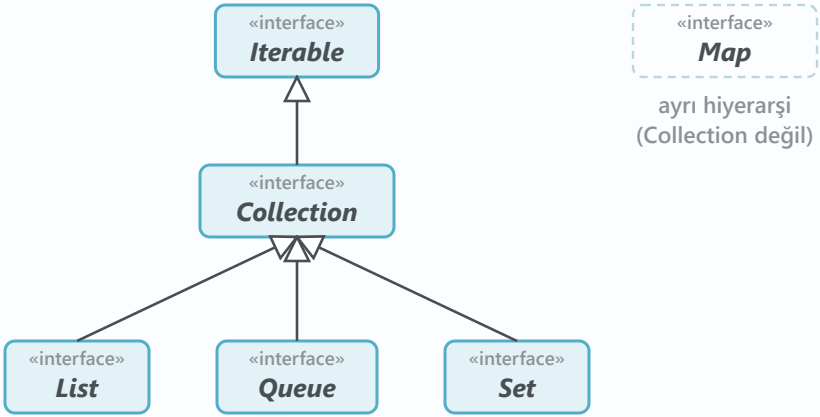
Geniş hiyerarşi

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

Hazır algoritmalar

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

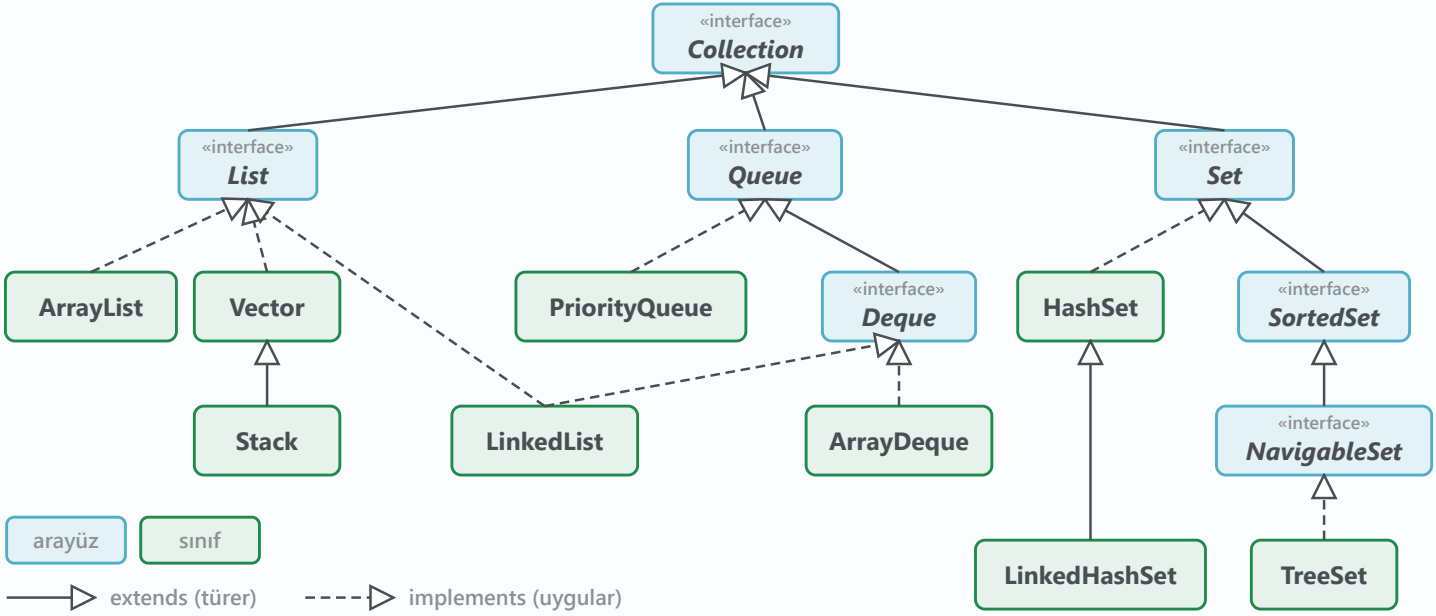
Koleksiyon Arayüzleri



- `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 uygular. Mavi kutular arayüz (yalnızca davranış), yeşil kutular `new` ile nesnesi oluşturulabilen sınıflardır.

KOLEKSİYON ÇERÇEVESİ

Çerçevenin Üç Unsuru

Arayüz (Interface)

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

Gerçekleme (Implementation)

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

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 **davranışı** tanımlar; sınıflar bu davranışı kendi metotlarıyla uygular.
- 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.
- Değişken ve parametre **arayüz tipinde** tanımlanır: gerçekleme tek satırda değiştirilebilir; 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() + " öğe)");
    }

    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 öğe)
Elma Armut Kiraz (3 öğe)
```

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);        // kutudan çıkarma → int
System.out.println(ilk + " " + n);
```

```
Ali 85
```

- `<>` içine **öğe tipi** yazılır: `List<String>` "String listesi" demektir.
- Yanlış tip derleme zamanında yakalanır; `get` sonucu için tip dönüşümü (cast) gerekmez.
- Sağdaki boş `<>` : elmas (diamond) işleci, 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 **kutudan çıkarma** (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 anahtarı (isim) bir değere (numara) eşler.
- `put(anahtar, değer)` ile ekleriz, `get(anahtar)` ile değeri buluruz.
- Listeyi `println` ile yazdırmak öğeleri `[ … ]` içinde gösterir.

KOLEKSİYON ÇERÇEVESİ

Collection Arayüzü Metotları

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

`E` : öğe 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.

List Arayüzü

- Öğeler **eklenme sırasına** göre tutulur: sıralı (ordered) ama sıralanmış (sorted) değil.
- **İndis** tabanlı erişim: ilk öğe 0, son öğe `size() - 1`.
- Aynı öğe 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
`add(E e)`	sona ekler
`add(int i, E e)`	`i` konumuna ekler
`get(int i)`	`i`'deki öğeyi verir
`set(int i, E e)`	`i`'deki öğeyi değiştirir
`remove(int i)`	`i`'deki öğeyi çıkarır
`remove(Object o)`	`o`'yu çıkarır
`indexOf(Object o)`	ilk konumu; yoksa −1
`size()`	öğe sayısı

ArrayList Sınıfı

- **Dinamik dizi**: öğeleri içerideki bir dizide, eklenme sırasıyla saklar.
- `AbstractList` sınıfından türetilmiştir ve `List` arayüzünü uygular.
- **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 kaydırdığı için yavaştır: O(n).

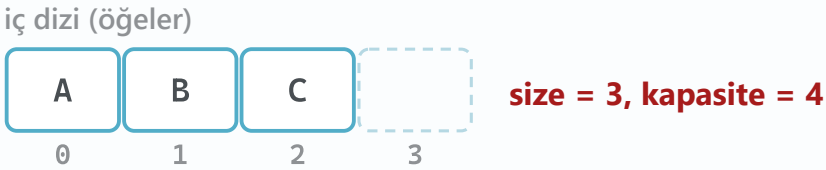
```
ArrayList<String> meyveler = new ArrayList<>();
meyveler.add("Elma");
meyveler.add("Armut");
System.out.println(meyveler.get(1));  // indeksle
```

```
Armut
```

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

LİSTELER

ArrayList Nasıl Büyür?

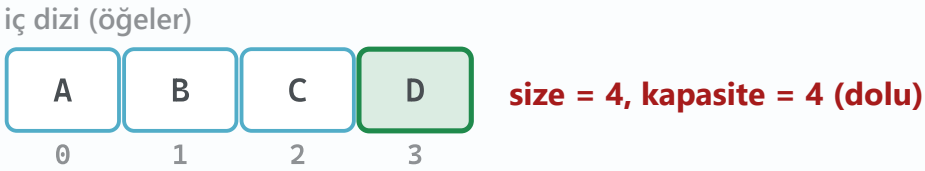


`new ArrayList<>(4)` ile başlangıç kapasitesi 4 olan bir liste düşünelim: içeride 4 hücrelik bir dizi var, 3'ü dolu.

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.

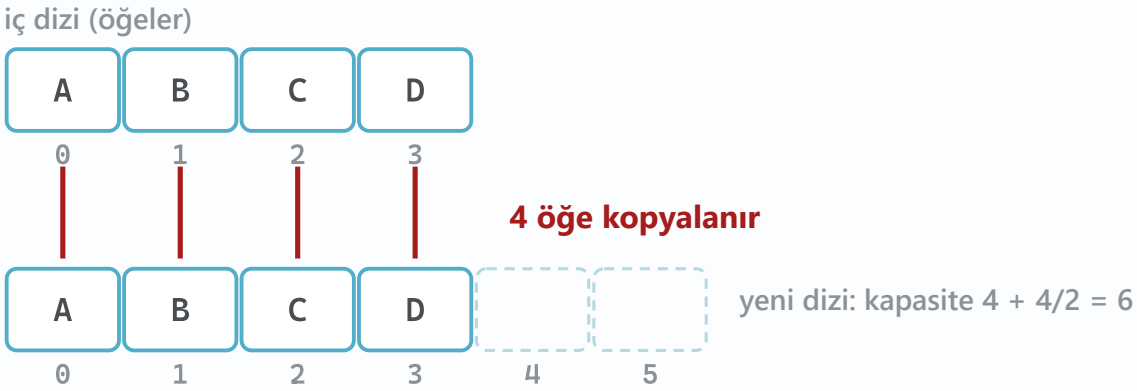
ArrayList Nasıl Büyür?



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

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.

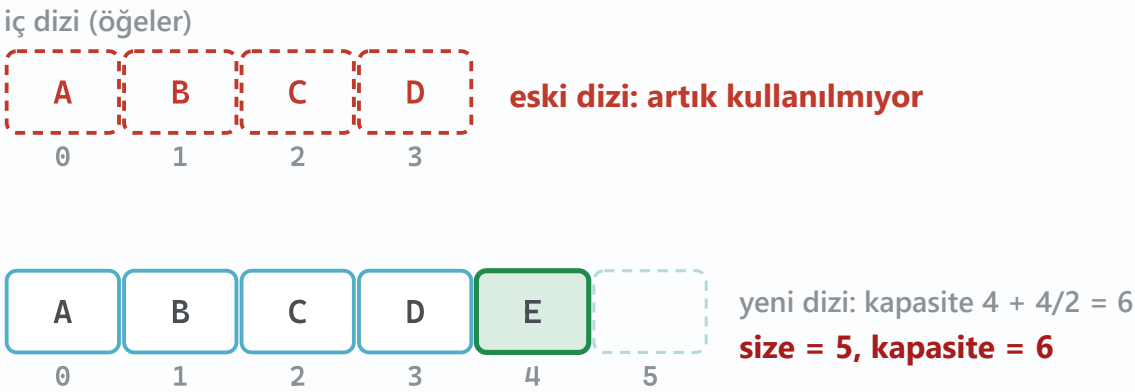
ArrayList Nasıl Büyür?



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

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 Nasıl Büyür?



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

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

```
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

boş liste

size = 0

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

Adım 1 / 10

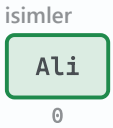
DEĞİŞKENLER

—

ÇIKTI

ArrayList Metotları: İzleme

```
1  ArrayList<String> isimler = new ArrayList<>();
▶  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());
```



size = 1

`add` öğeyi sona ekler: Ali 0. indiste.

Adım 2 / 10

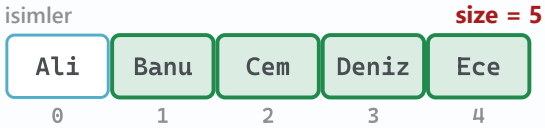
DEĞİŞKENLER

—

ÇIKTI

ArrayList Metotları: İzleme

```
1  ArrayList<String> isimler = new ArrayList<>();
2  isimler.add("Ali");
▶  isimler.add("Banu");
▶  isimler.add("Cem");
▶  isimler.add("Deniz");
▶  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");
▶  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());
```



Liste yazdırılır: öğeler eklenme sırasıyla.

Adım 4 / 10

DEĞİŞKENLER

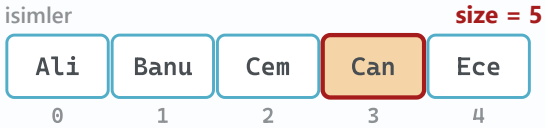
—

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());
```



`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);
▶ isimler.remove(4);            // 4. indeksi sil
▶ 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());
```



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

Adım 6 / 10

DEĞİŞKENLER

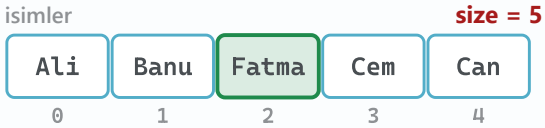
—

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());
```



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

Adım 7 / 10

DEĞİŞKENLER

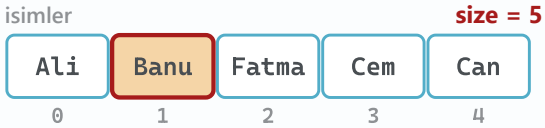
—

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);
▶ int indeks = isimler.indexOf("Banu");
▶ 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());
```



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

Adım 8 / 10

DEĞİŞKENLER

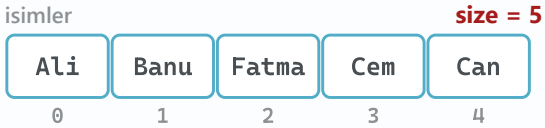
`indeks = 1`

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);
▶  boolean varMi = isimler.contains("Ece");
▶  System.out.println("Ece var mı: " + varMi);
18 System.out.println("boyut: " + isimler.size());
```



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

Adım 9 / 10

DEĞİŞKENLER

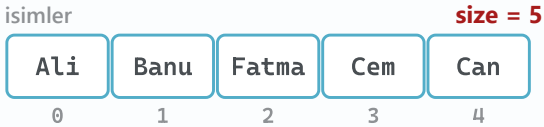
`indeks = 1` `varMi = false`

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);
▶  System.out.println("boyut: " + isimler.size());
```



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
```

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);
```



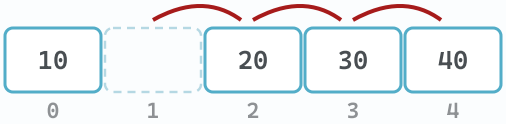
Başlangıç: `[10, 20, 30, 40]`. Şimdi `add(1, 15)` çalışacak.

Adım 1 / 5

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

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.


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

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]

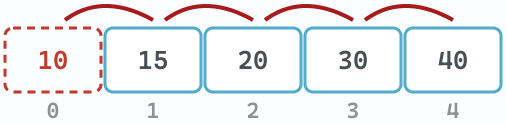
15 boşalan hücreye yazılır. *n* öğeli listede en kötü durumda *n* kaydırma: O(n).

Adım 3 / 5

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

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 öğe sola

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


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

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]

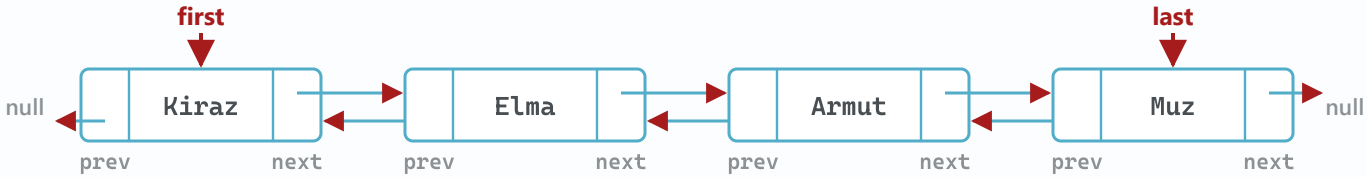
Sonuç. Sondan silmede kaydırma gerekmez: O(1). Baştan/ortadan silmede O(n).

Adım 5 / 5

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

LinkedList Sınıfı

- **Çift yönlü bağlı liste**: öğeler **düğümlerde** (node) saklanır; her düğüm değeri ve önceki (prev) ile sonraki (next) düğümün referansını tutar.
- `AbstractSequentialList` sınıfından türetilmiştir; `List` ve `Deque` arayüzlerini uygular.
- **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.

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
`addFirst(e)`	başa ekler
`addLast(e)`	sona ekler
`getFirst()`	ilk öğeyi verir
`getLast()`	son öğeyi verir
`removeFirst()`	ilk öğeyi çıkarıp verir
`removeLast()`	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
```

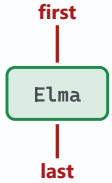
Boş liste: `first` ve `last` referansları `null`. Kapasite diye bir şey yok.

Adım 1 / 10

ÇIKTI

LinkedList: Düğümler ve Bağlar

```
1  LinkedList<String> meyveler = new LinkedList<>();
▶  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);
```



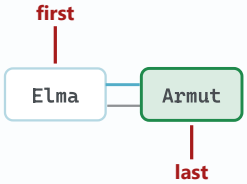
Elma için bir düğüm oluşur; tek düğüm hem ilk hem son.

Adım 2 / 10

ÇIKTI

LinkedList: Düğümler ve Bağlar

```
1  LinkedList<String> meyveler = new LinkedList<>();
2  meyveler.add("Elma");
▶  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);
```



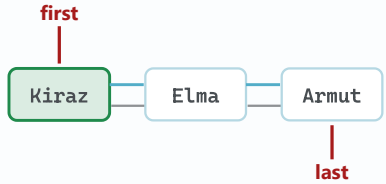
Armut sona bağlanır: Elma.next → Armut, Armut.prev → Elma; `last` güncellenir.

Adım 3 / 10

ÇIKTI

LinkedList: Düğümler ve Bağlar

```
1  LinkedList<String> meyveler = new LinkedList<>();
2  meyveler.add("Elma");
3  meyveler.add("Armut");
▶  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);
```



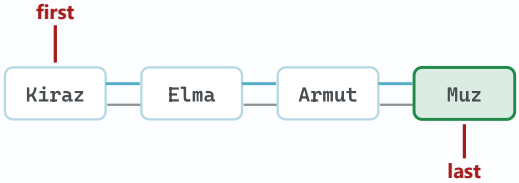
`addFirst` : Kiraz başa; yalnızca birkaç referans değişir, öğe kaydırılmaz: O(1).

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");
▶  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);
```



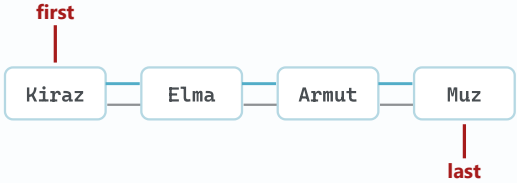
`addLast` : Muz sona, yine O(1).

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");
▶  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` bağlarıyla gezerek yazar.

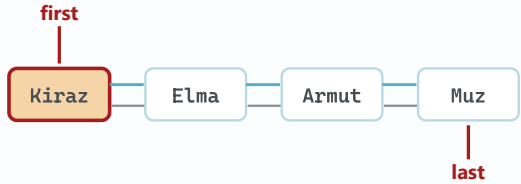
Adım 6 / 10

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);
▶  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.

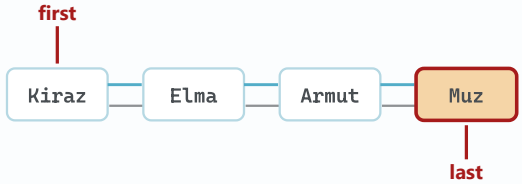
Adım 7 / 10

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());
▶  System.out.println(meyveler.getLast());
9  meyveler.removeFirst();
10 System.out.println(meyveler);
```



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

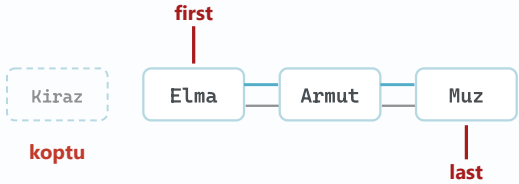
Adım 8 / 10

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());
▶  meyveler.removeFirst();
10 System.out.println(meyveler);
```



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

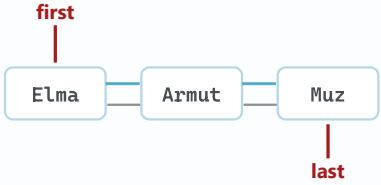
Adım 9 / 10

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);
```



`[Elma, Armut, Muz]` yazılır.

Adım 10 / 10

ÇIKTI

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

ArrayList mi, LinkedList mi?

İşlem	ArrayList	LinkedList
`get(i)`, `set(i, e)`	**O(1)**	O(n): düğüm düğüm yürür
Sona ekleme `add(e)`	**O(1)** ortalama (arada bir kopyalama)	**O(1)**
Başa 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 doğru seçim **ArrayList**'tir. Öğe ekleme ve çıkarma işlemleri sık ve özellikle listenin **uçlarında** ise `LinkedList` (ya da kuyruk/yığın için `ArrayDeque`) tercih edilebilir.

Vector Sınıfı

- `List` arayüzünü uygulayan, genişletilebilir bir dinamik dizidir.
- Belirtilen (varsayılan 10) kapasiteyle başlar; doldukça kapasitesi **iki katına** çıkar.
- Metotları senkronizedir (synchronized): çoklu iş parçacıklı ortamda güvenle 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 öğe
System.out.println(v.size() + " / " + v.capacity());
```

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

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

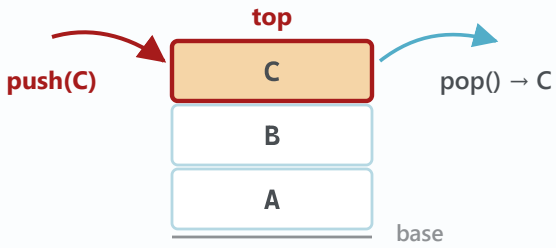
`addElement(E obj)`	`elementAt(int i)`	`removeElement(Object obj)`	`size()`	`capacity()`
öğe ekler	`i`'deki öğeyi verir	öğeyi çıkarır	öğe sayısı	mevcut kapasite

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 uygulayan `ArrayDeque` (veya `LinkedList`) önerilir.

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

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



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());
```



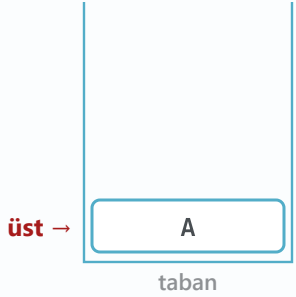
Boş bir yığın oluşur.

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());
```



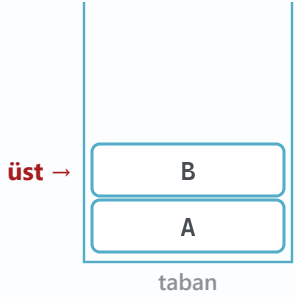
`push("A")` : A en üste konur.

Adım 2 / 9

ÇIKTI

Stack: İzleme

```
1 Stack<String> yigin = new Stack<>();
2 yigin.push("A");
▶ 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());
```



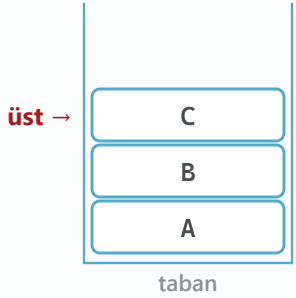
`push("B")` : B, A'nın üstüne.

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());
```



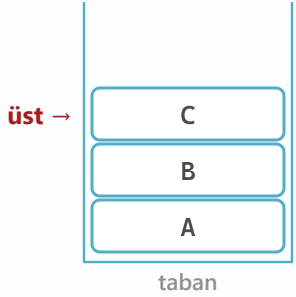
`push("C")` : en üstte artık C var.

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]`.

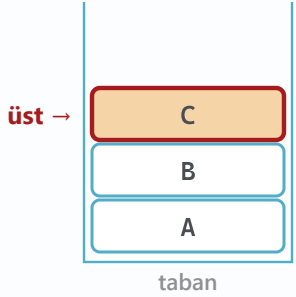
Adım 5 / 9

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());
```



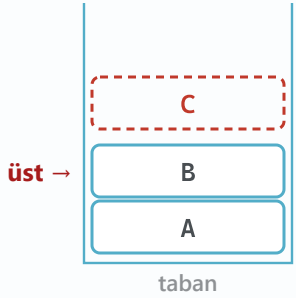
`peek()` en üsttekini (C) döndürür ama çıkarmaz.

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());
▶ 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.

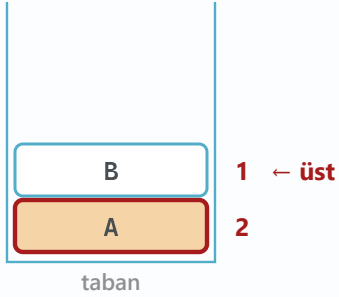
Adım 7 / 9

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.

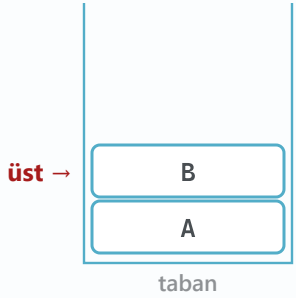
Adım 8 / 9

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 öğe var, sonuç `false` .

Adım 9 / 9

ÇIKTI

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

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();
```

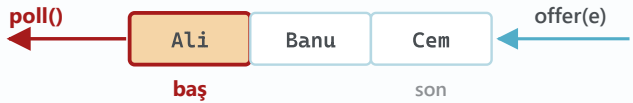
ÇIKTI

```
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 öğe silmek gerekiyorsa `Iterator` kullanılır.

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)	`add(e)`	`offer(e)` → `false`
Çıkar (baştan)	`remove()`	`poll()` → `null`
Bak (baştakine)	`element()`	`peek()` → `null`

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

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 `Queue` tipinde, nesne `LinkedList`.

Adım 1 / 11

ÇIKTI

Queue: İzleme

```
1  Queue<String> kuyruk = new LinkedList<>();
▶  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());
```

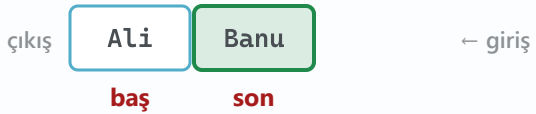


`offer("Ali")` : Ali kuyruğa girer; hem baş hem son o.

ÇIKTI

Queue: İzleme

```
1  Queue<String> kuyruk = new LinkedList<>();
2  kuyruk.offer("Ali");
▶  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());
```



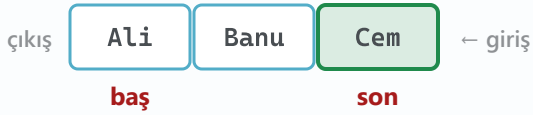
Banu sona girer.

Adım 3 / 11

ÇIKTI

Queue: İzleme

```
1  Queue<String> kuyruk = new LinkedList<>();
2  kuyruk.offer("Ali");
3  kuyruk.offer("Banu");
▶  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());
```



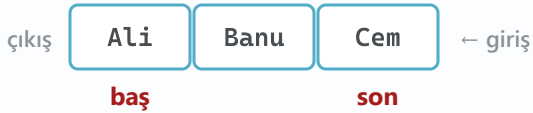
Cem sona girer.

Adım 4 / 11

ÇIKTI

Queue: İzleme

```
1  Queue<String> kuyruk = new LinkedList<>();
2  kuyruk.offer("Ali");
3  kuyruk.offer("Banu");
4  kuyruk.offer("Cem");
▶  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.

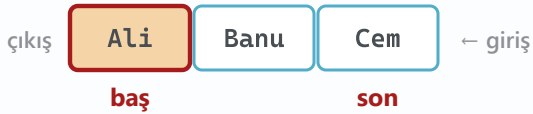
Adım 5 / 11

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);
▶  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.

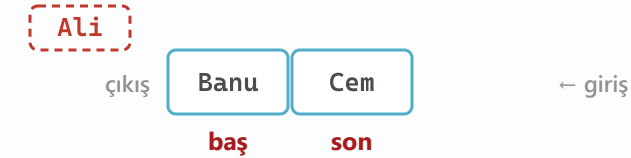
Adım 6 / 11

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());
▶  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());
```



`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());
▶  System.out.println(kuyruk.poll());
9  System.out.println(kuyruk.size());
10 System.out.println(kuyruk.poll());
11 System.out.println(kuyruk.poll());
```



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());
▶  System.out.println(kuyruk.size());
10 System.out.println(kuyruk.poll());
11 System.out.println(kuyruk.poll());
```



`size()` : kuyrukta 1 öğe kaldı.

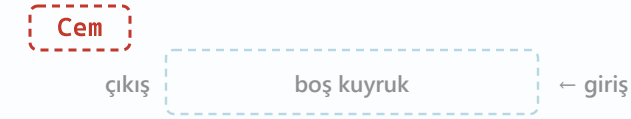
Adım 9 / 11

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());
▶  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

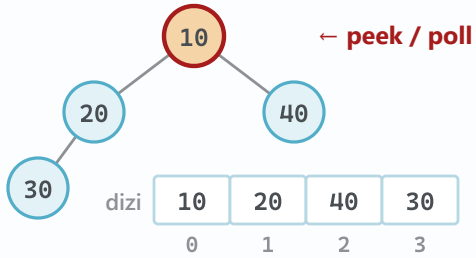
ÇIKTI

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

PriorityQueue Sınıfı

- **Öncelikli kuyruk** (priority queue): çıkan öğe her zaman **en öncelikli** olandır, eklenme sırası değil.
- Varsayılan öncelik doğal sıradır: en küçük öğe önce çıkar. Tersi 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 çizelgeleme, acil servis sırası, en kısa yol algoritmaları.

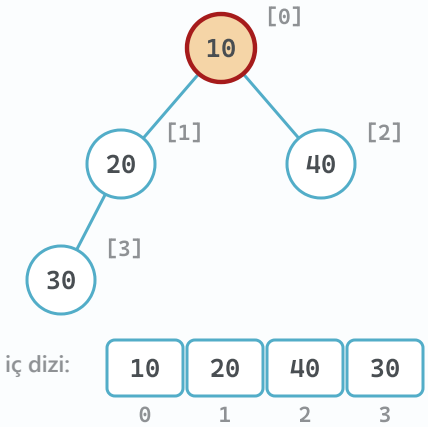
Metot	Görevi
`add(E e)`	öğe ekler
`poll()`	en öncelikliyi çıkarır ve döndürür
`peek()`	en öncelikliyi döndürür, çıkarmaz
`remove(Object o)`	belirtilen öğeyi çıkarır
`size()`	öğe sayısı



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

```
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)



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

Adım 1 / 12

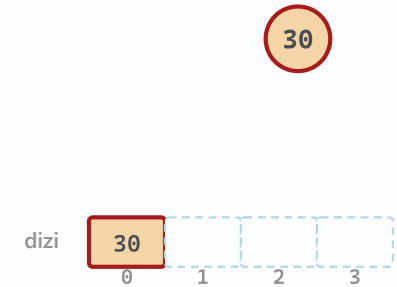
DEĞİŞKENLER

—

ÇIKTI

PriorityQueue: Yığın Kuralı Adım Adım

```
1  PriorityQueue<Integer> pq = new PriorityQueue<>();
▶  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 köke (indis 0) yerleşir.

Adım 2 / 12

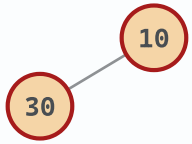
DEĞİŞKENLER

—

ÇIKTI

PriorityQueue: Yığın Kuralı Adım Adım

```
1  PriorityQueue<Integer> pq = new PriorityQueue<>();
2  pq.add(30);
▶  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 olan 30'dan küçük olduğu için **yukarı çıkar**, yer değiştirirler. Kök artık 10.

Adım 3 / 12

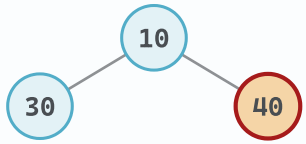
DEĞİŞKENLER

—

ÇIKTI

PriorityQueue: Yığın Kuralı Adım Adım

```
1  PriorityQueue<Integer> pq = new PriorityQueue<>();
2  pq.add(30);
3  pq.add(10);
▶  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 olan 10'dan büyük, yerinde kalır.

Adım 4 / 12

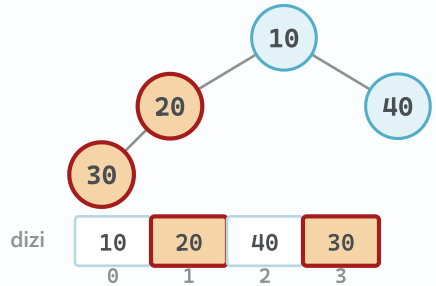
DEĞİŞKENLER

—

ÇIKTI

PriorityQueue: Yığın Kuralı Adım Adım

```
1  PriorityQueue<Integer> pq = new PriorityQueue<>();
2  pq.add(30);
3  pq.add(10);
4  pq.add(40);
▶  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 olan 30'dan küçük → yer değiştirir; 10'dan büyük → durur.

Adım 5 / 12

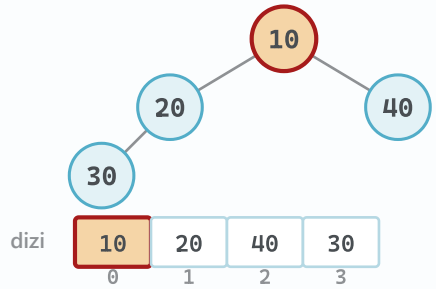
DEĞİŞKENLER

—

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);
▶  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 öğe: 10. Kuyruk değişmez; O(1).

Adım 6 / 12

DEĞİŞKENLER

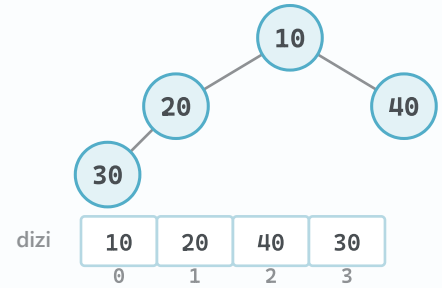
—

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());
▶  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

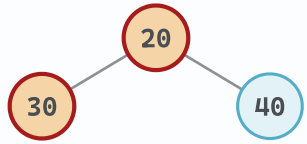
—

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()) {
▶      System.out.print(pq.poll() + " ");
10 }
11 System.out.println();
```





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

Adım 8 / 12

DEĞİŞKENLER

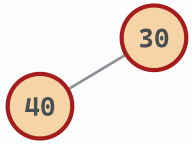
`x = 10`

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()) {
▶        System.out.print(pq.poll() + " ");
10 }
11 System.out.println();
```





20 çıkar; 40 köke gelir, 30 ile yer değiştirir.

Adım 9 / 12

DEĞİŞKENLER

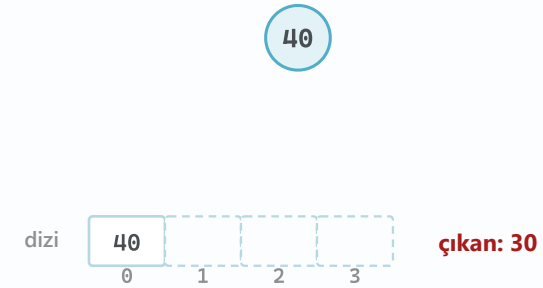
`x = 20`

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()) {
▶      System.out.print(pq.poll() + " ");
10 }
11 System.out.println();
```



30 çıkar; tek öğe 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()) {
▶      System.out.print(pq.poll() + " ");
10 }
11 System.out.println();
```

(boş)



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)



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());
}
```

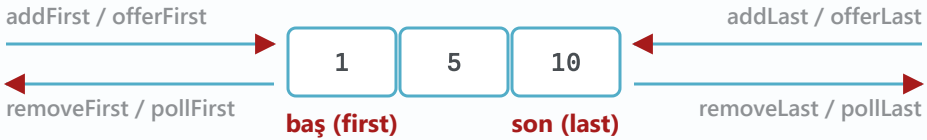
```
Arıza(1)
Toplantı(2)
Rapor(3)
```

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

```
40
```

Deque Arayüzü

- **Deque** (Double-Ended Queue, "dek" okunur): hem baştan hem sondan öğe 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
`addFirst(e)`	başa ekler	`addLast(e)`	sona ekler
`removeFirst()`	baştakini çıkarır, döndürür	`removeLast()`	sondakini çıkarır, döndürür
`getFirst()`	baştakini döndürür	`getLast()`	sondakini döndürür

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

ArrayDeque Sınıfı

- Çift uçlu kuyruk yapısını **dinamik (dairesel) bir diziyle** uygular.
- 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
`pollFirst()` / `pollLast()`	baştakini / sondakini çıkarır, döndürür
`peekFirst()` / `peekLast()`	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

```
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());
```



Boş bir `ArrayDeque`.

Adım 1 / 9

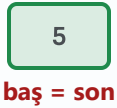
DEĞİŞKENLER

—

ÇIKTI

ArrayDeque: İzleme

```
1 ArrayDeque<Integer> sayilar = new ArrayDeque<>();
▶ 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(5)` : 5 başa eklenir (tek öğe).

Adım 2 / 9

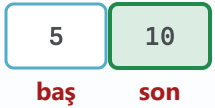
DEĞİŞKENLER

—

ÇIKTI

ArrayDeque: İzleme

```
1 ArrayDeque<Integer> sayilar = new ArrayDeque<>();
2 sayilar.addFirst(5);
▶ 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

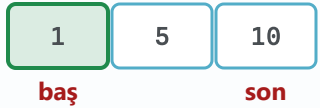
DEĞİŞKENLER

—

ÇIKTI

ArrayDeque: İzleme

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



`pollLast()` sondaki 10'u çıkarır.

Adım 7 / 9

DEĞİŞKENLER

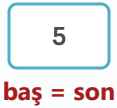
`ilkSayi = 1` `sonSayi = 10`

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());
```



İki değer yazdırılır.

Adım 8 / 9

DEĞİŞKENLER

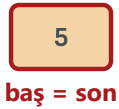
`ilkSayi = 1` `sonSayi = 10`

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());
```



`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` eklemez, `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
`add(E e)`	öğeyi kümeye ekler
`remove(Object o)`	öğeyi kümeden çıkarır
`contains(Object o)`	öğe kümede var mı?
`size()`	öğe 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 **sırasız** saklar; `AbstractSet`'ten türetilmiştir ve `Set`'i uygular.
- 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

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

HashSet ve Hashing

- Tekil öğeleri **sırasız** saklar; `AbstractSet`'ten türetilmiştir ve `Set`'i uygular.
- 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.



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

HashSet ve Hashing

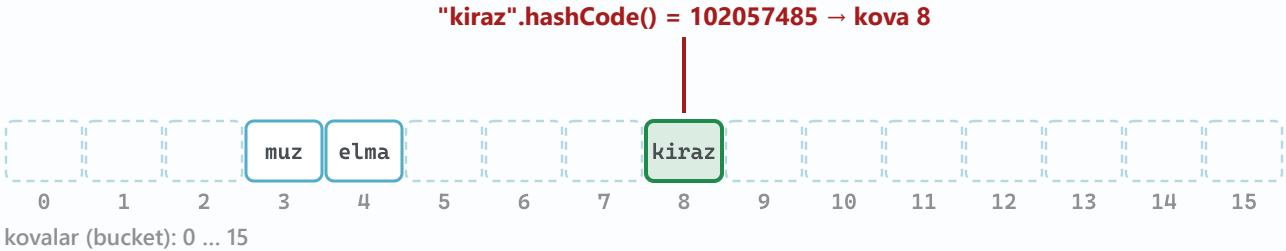
- Tekil öğeleri **sırasız** saklar; `AbstractSet`'ten türetilmiştir ve `Set`'i uygular.
- Öğenin `hashCode()` değerinden bir **kova** (bucket) numarası hesaplanır; öğe doğrudan o kovaya gider: ekleme ve arama ortalama O(1).
- Depolama sırası garanti edilmez: gezinme kovaların sırasıyla yapılır.



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

HashSet ve Hashing

- Tekil öğeleri **sırasız** saklar; `AbstractSet`'ten türetilmiştir ve `Set`'i uygular.
- 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.



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

HashSet ve Hashing

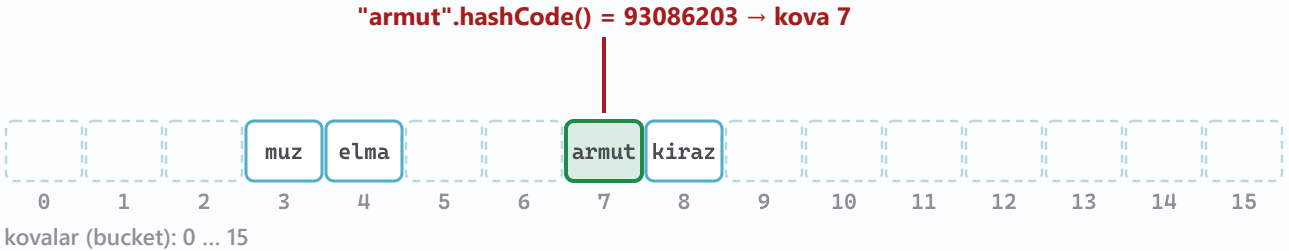
- Tekil öğeleri **sırasız** saklar; `AbstractSet`'ten türetilmiştir ve `Set`'i uygular.
- Öğenin `hashCode()` değerinden bir **kova** (bucket) numarası hesaplanır; öğe doğrudan o kovaya gider: ekleme ve arama ortalama O(1).
- Depolama sırası garanti edilmez: gezinme kovaların sırasıyla yapılır.



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

HashSet ve Hashing

- Tekil öğeleri **sırasız** saklar; `AbstractSet`'ten türetilmiştir ve `Set`'i uygular.
- 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.



`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 listeyle 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
`add(E e)`	öğe ekler
`remove(Object o)`	öğeyi çıkarır
`contains(Object o)`	öğe var mı?
`size()`	öğe sayısı
`clear()`	tüm öğeleri çıkarır

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

```java
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

```java
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

```
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



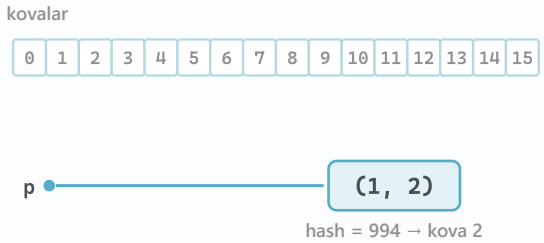
`HashSet` içte 16 kovalı bir hash tablosu kullanır. `Nokta`'nın `equals` ve `hashCode` metotları `x` ve `y`'den hesaplanır.

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());
```



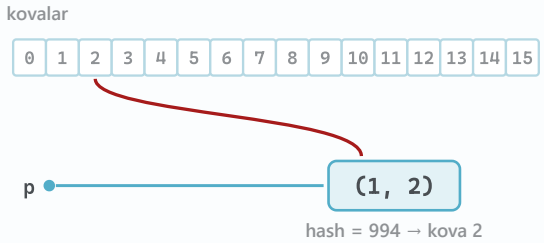
`Objects.hash(1, 2)` = 994. Kova numarası hash'in son 4 bitidir: 994 → kova 2.

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());
```



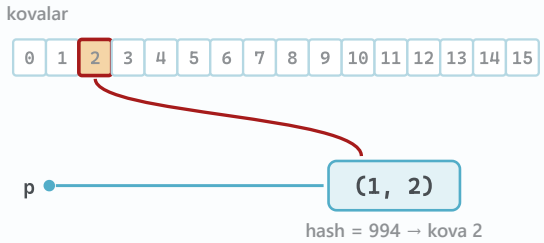
`add(p)` : nesne 2 numaralı kovaya yerleşir.

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);
▶ 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());
```



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

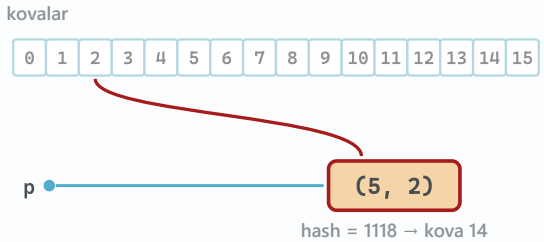
Adım 4 / 9

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));
▶ 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());
```



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

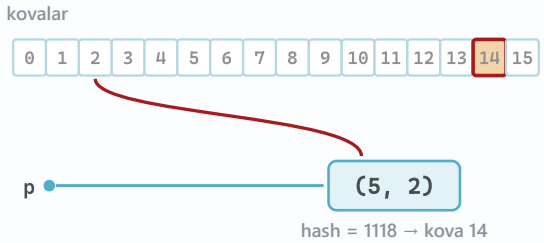
Adım 5 / 9

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
▶ 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());
```



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

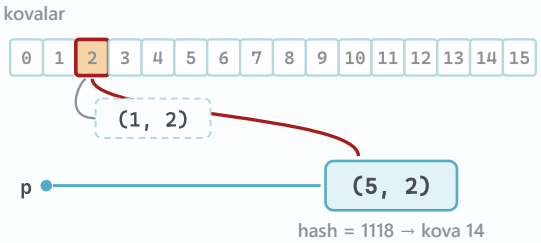
Adım 6 / 9

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));
▶ System.out.println(kume.contains(new Nokta(1, 2)));
8 kume.add(p);
9 System.out.println(kume.size());
```



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

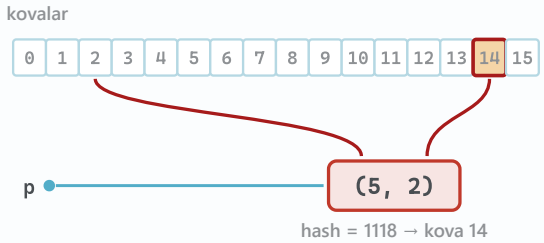
Adım 7 / 9

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)));
▶ kume.add(p);
9 System.out.println(kume.size());
```



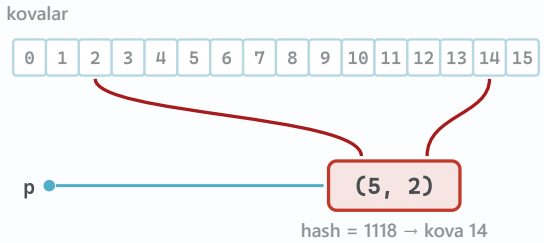
`add(p)` : kova 14 boş göründüğü için **aynı nesne ikinci kez** eklenir.

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());
```



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

Adım 9 / 9

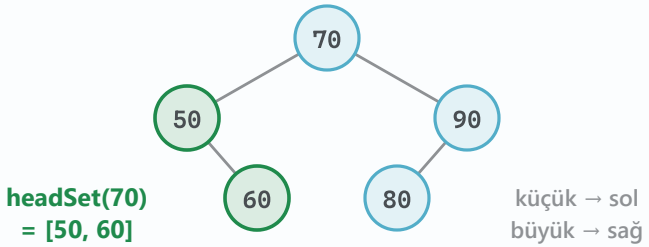
ÇIKTI

```
true
false
false
2
```

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 uygular.
- İçte kırmızı-siyah ağaç: ekleme, silme, arama O(log n).

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



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.

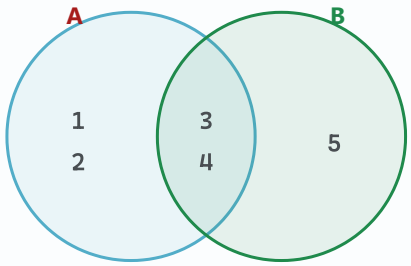
Küme İşlemleri

```
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 ∪ B = [1, 2, 3, 4, 5]
A ∩ B = [3, 4]
A − B = [1, 2]
true
```



A ∩ B

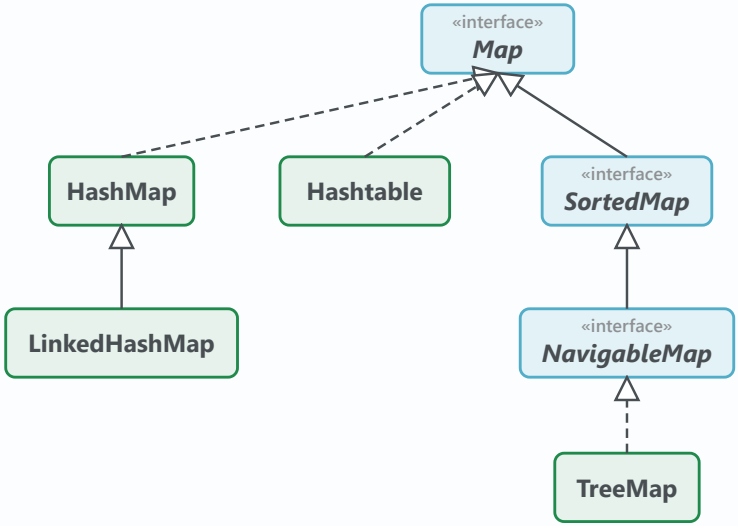
İşlem	Metot
birleşim ∪	`addAll`
kesişim ∩	`retainAll`
fark −	`removeAll`
alt küme ⊆	`containsAll`

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 anlamı (değer) bulunur; aynı kelime iki kez yer almaz.



Map Metotları

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

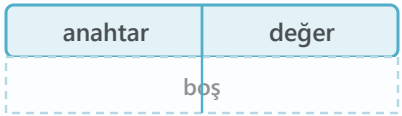
for (String isim : notlar.keySet()) {
    System.out.print(isim + " ");
}
System.out.println();
for (Integer not : notlar.values()) {
    System.out.print(not + " ");
}
System.out.println();
for (Map.Entry<String, Integer> g : notlar.entrySet()) {
    System.out.println(g.getKey() + " → " + g.getValue());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

```
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<>();
for (String k : metin.split(" ")) {
    if (sayac.containsKey(k)) {
        sayac.put(k, sayac.get(k) + 1);
    } else {
        sayac.put(k, 1);
    }
}
System.out.println(sayac);
```



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<>();
▶  for (String k : metin.split(" ")) {
▶      if (sayac.containsKey(k)) {
5          sayac.put(k, sayac.get(k) + 1);
6      } else {
▶          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<>();
▶  for (String k : metin.split(" ")) {
▶      if (sayac.containsKey(k)) {
5          sayac.put(k, sayac.get(k) + 1);
6      } else {
▶          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<>();
▶  for (String k : metin.split(" ")) {
▶      if (sayac.containsKey(k)) {
▶          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<>();
▶  for (String k : metin.split(" ")) {
▶      if (sayac.containsKey(k)) {
5          sayac.put(k, sayac.get(k) + 1);
6      } else {
▶          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<>();
▶ for (String k : metin.split(" ")) {
▶     if (sayac.containsKey(k)) {
▶         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<>();
▶  for (String k : metin.split(" ")) {
▶      if (sayac.containsKey(k)) {
▶          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 öğe 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 uygular.
- Anahtarlar doğal sıraya ya da verilen `Comparator` 'a göre dizilir.
- İçte **kırmızı-siyah ağaç** (red-black tree): `get` / `put` O(log n).

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

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}
```

Map Sınıflarını Karşılaştırma

	HashMap	LinkedHashMap	TreeMap	Hashtable
Gezinme sırası	garanti yok	eklenme sırası	anahtar sıralı	garanti yok
`get` / `put`	O(1) ortalama	O(1) ortalama	O(log n)	O(1) ortalama, kilitli
`null` anahtar	bir tane	bir tane	hayır	hayır
`null` 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.

YİNELEYİCİ

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
`hasNext()`	bir sonraki öğe var mı?
`next()`	sonraki öğeyi döndürür, imleci ilerletir
`remove()`	son `next()` 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);
Iterator<Integer> iterator = liste.iterator();
while (iterator.hasNext()) {
    Integer sayi = iterator.next();
    // öğe burada işlenir
    System.out.println(sayi);
}
```



Liste hazırlanır: `[3, 8, 5]`.

Adım 1 / 9

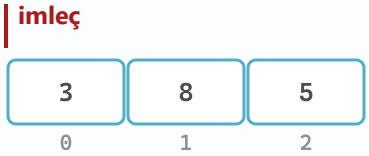
DEĞİŞKENLER

—

ÇIKTI

Iterator: İzleme

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



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

Adım 2 / 9

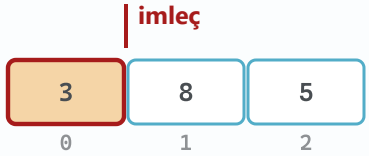
DEĞİŞKENLER

—

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      // öğe burada işlenir
9      System.out.println(sayi);
10 }
```



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

Adım 3 / 9

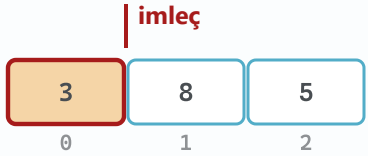
DEĞİŞKENLER

`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      // öğe burada işlenir
▶      System.out.println(sayi);
10 }
```



Öğe yazdırılır.

Adım 4 / 9

DEĞİŞKENLER

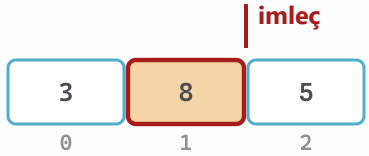
`sayi = 3`

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      // öğe burada işlenir
9      System.out.println(sayi);
10 }
```



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

Adım 5 / 9

DEĞİŞKENLER

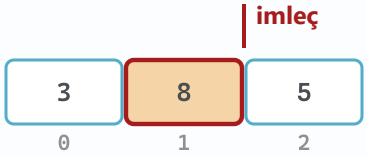
`sayi = 8`

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      // öğe burada işlenir
▶      System.out.println(sayi);
10 }
```



Öğe yazdırılır.

Adım 6 / 9

DEĞİŞKENLER

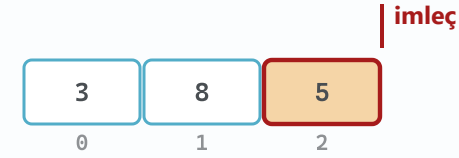
`sayi = 8`

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      // öğe burada işlenir
9      System.out.println(sayi);
10 }
```



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

Adım 7 / 9

DEĞİŞKENLER

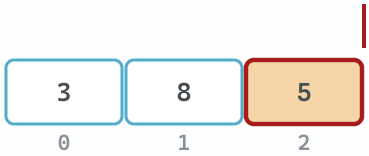
`sayi = 5`

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      // öğe burada işlenir
▶      System.out.println(sayi);
10 }
```



Öğe yazdırılır.

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      // öğe burada işlenir
9      System.out.println(sayi);
10 }
```



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

Adım 9 / 9

DEĞİŞKENLER

`sayi = 5`

ÇIKTI

```
3
8
5
```

YİNELEYİCİ

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.

YİNELEYİCİ · ADIM ADIM KOD İZLEME

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)



size = 3

modCount = 0

Liste 3 öğeli; `modCount` (yapısal değişiklik sayacı) 0.

Adım 1 / 8

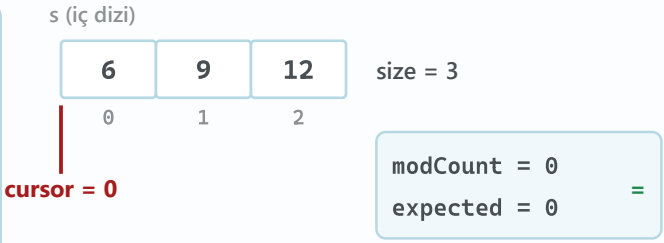
DEĞİŞKENLER

`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);
▶ for (Integer x : s) {      // gizli Iterator
4     if (x % 2 == 0) {
5         s.remove(x);
6     }
7 }
8 System.out.println(s);
```



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

Adım 2 / 8

DEĞİŞKENLER

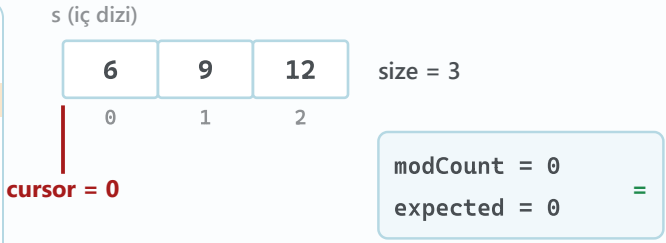
`s = [6, 9, 12]` `modCount = 0`

ÇIKTI

YİNELEYİCİ · ADIM ADIM KOD İZLEME

for-each İçinde Silmek: İstisna Nereden Gelir?

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



`hasNext()` : cursor (0) ≠ size (3) → true.

Adım 3 / 8

DEĞİŞKENLER

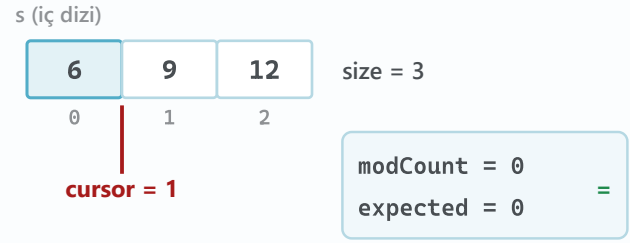
`s = [6, 9, 12]` `modCount = 0` `hasNext = true`

ÇIKTI

YİNELEYİCİ · ADIM ADIM KOD İZLEME

for-each İçinde Silmek: İstisna Nereden Gelir?

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



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

Adım 4 / 8

DEĞİŞKENLER

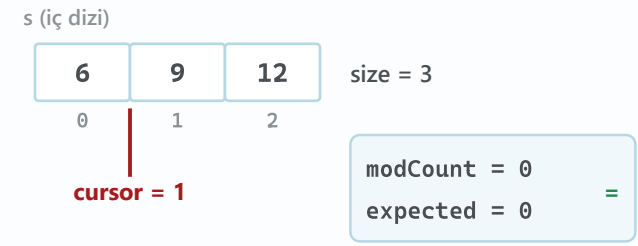
`s = [6, 9, 12]` `modCount = 0` `x = 6`

ÇIKTI

YİNELEYİCİ · ADIM ADIM KOD İZLEME

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
▶     if (x % 2 == 0) {
5         s.remove(x);
6     }
7 }
8 System.out.println(s);
```



6 çift.

Adım 5 / 8

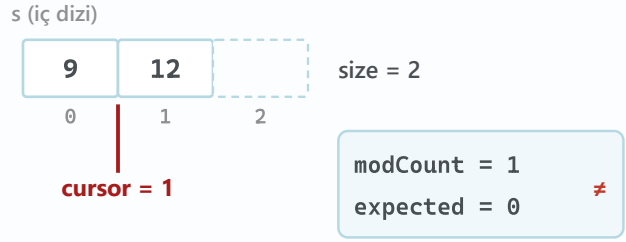
DEĞİŞKENLER

`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) {
▶         s.remove(x);
6     }
7 }
8 System.out.println(s);
```



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

Adım 6 / 8

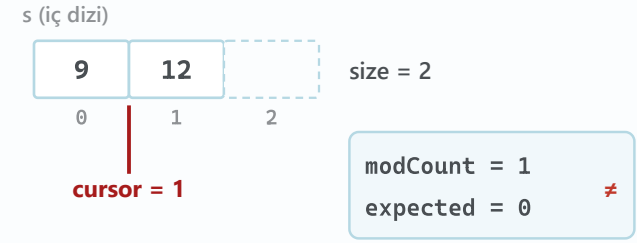
DEĞİŞKENLER

`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);
▶ for (Integer x : s) {      // gizli Iterator
4     if (x % 2 == 0) {
5         s.remove(x);
6     }
7 }
8 System.out.println(s);
```



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

Adım 7 / 8

DEĞİŞKENLER

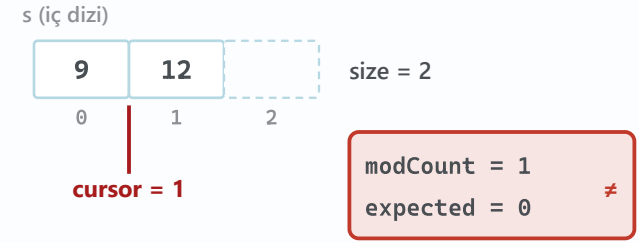
`s = [9, 12]` `modCount = 1` `hasNext = true`

ÇIKTI

YİNELEYİCİ · ADIM ADIM KOD İZLEME

for-each İçinde Silmek: İstisna Nereden Gelir?

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



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
```

ALGORİTMALAR

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; elle döngü yazmak yerine bunlar kullanılır.
- Doğru algoritmayı seçmek performansı önemli ölçüde etkiler.

Sıralama

`sort`, `reverse`, `reverseOrder`

Karıştırma

`shuffle`

Arama

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

Kompozisyon

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

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
`sort(list)`	doğal sıraya göre; öğeler `Comparable` olmalı
`sort(list, c)`	verilen karşılaştırıcıya (`Comparator`) göre
`reverse(list)`	listeyi ters sıraya çevirir

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

`Collections.sort` : **TimSort**, O(n log n), **kararlı** (eşitlerin sırası korunur); `Arrays.sort` ilkellerde Dual-Pivot Quicksort.

Comparable: Doğal Sıra

```java
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;
    }
}
```

```java
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 uygular ve `compareTo`'yu yazar.
- Sonuç: negatif → `this` önce; 0 → eşit; pozitif → `diger` önce.
- `Integer.compare` çıkarma yapmaktan (`puan - diger.puan`) güvenlidir: taşma 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: *n* − 1 yer değiştirme, 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.

ALGORİTMALAR

Loto Örneği

```java
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.

ALGORİTMALAR

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.**

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

```
3
-4
3
```

7, üç karşılaştırmada indis 3'te bulunur. 6 yoktur; girmesi gereken yer (ekleme noktası) 3'tür ve dönüş −(3) − 1 = −4 olur.

Bulunamazsa dönüş **negatiftir**: −(ekleme noktası) − 1. Liste sıralı değilse sonuç **tanımsızdır**. 1 milyon öğede doğrusal arama 1 milyona kadar, ikili arama en fazla 20 karşılaştırma yapar.

Kompozisyon Algoritmaları

Metot	Görevi
`copy(hedef, kaynak)`	kaynağın öğelerini hedefin başına kopyalar; `hedef.size() ≥ kaynak.size()` olmalı
`fill(liste, d)`	tüm öğeleri `d` yapar
`reverse(liste)`	öğelerin sırasını tersine çevirir
`swap(liste, i, j)`	`i` ve `j` konumlarını değiştirir
`max` / `min`	en büyük / en küçük öğe
`frequency(c, o)`	`o`'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"));
Collections.copy(hedef, isimler);
System.out.println(hedef);
Collections.reverse(hedef);
System.out.println(hedef);
Collections.swap(hedef, 0, 2);
System.out.println(hedef);
Collections.fill(hedef, "Java");
System.out.println(hedef);
```



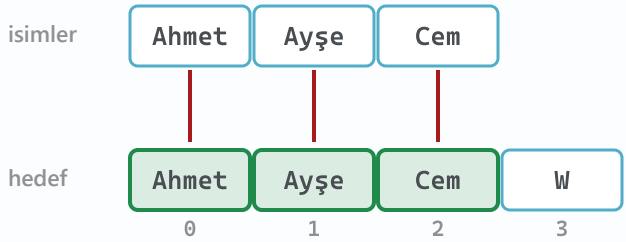
`Arrays.asList` sabit boyutlu bir liste; `hedef` ise 4 öğeli bir `ArrayList`.

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.

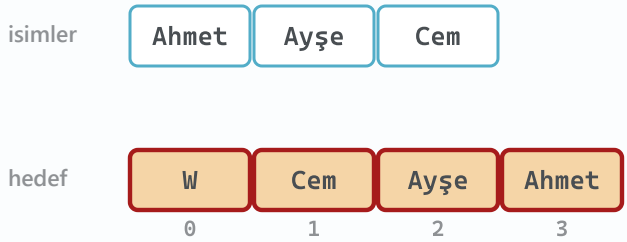
Adım 2 / 5

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);
▶  Collections.reverse(hedef);
▶  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);
```



`reverse` : sıra tersine döner.

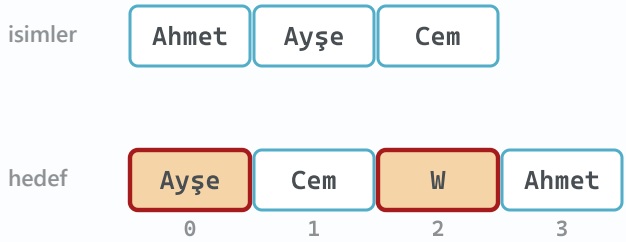
Adım 3 / 5

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);
```



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

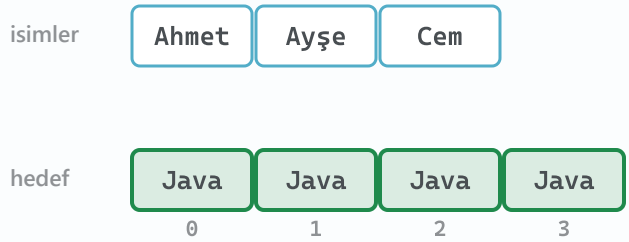
Adım 4 / 5

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);
```



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

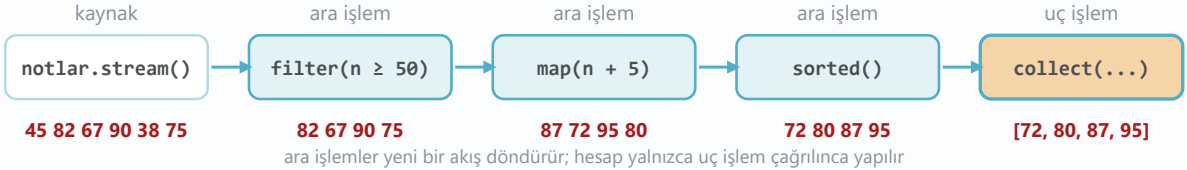
Adım 5 / 5

ÇIKTI

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

Akış API'si (`java.util.stream`)

- Java 8 ile gelen `java.util.stream`, bir veri dizisini **işlem hattı** (pipeline) olarak işler: "ne yapılacağını" söyleriz, döngüyü kütüphane kurar.
- Buradaki akış, `InputStream` gibi G/Ç akışlarından **farklıdır**: veri saklamaz, bir kaynağın (liste, dizi, dosya satırları) üzerinden geçer ve kaynağı değiştirmez.




```
List<Integer> notlar = Arrays.asList(45, 82, 67, 90, 38, 75);
List<Integer> gecenler = notlar.stream()
    .filter(n -> n >= 50)
    .map(n -> n + 5)
    .sorted()
    .collect(Collectors.toList());
System.out.println(gecenler);
```

ÇIKTI

```
[72, 80, 87, 95]
```

Java 16+ sürümünde son satır kısaca `.toList()` yazılabilir. Gereken importlar: `java.util.*`, `java.util.stream.Collectors`.

AKIŞ API'Sİ

Ara ve Uç İşlemler

Ara işlem	Görevi
`filter(p)`	koşulu sağlayanları geçirir
`map(f)`	her öğeyi dönüştürür
`flatMap(f)`	iç içe akışları düzleştirir
`sorted()`	sıralar (tüm öğeleri bekler)
`distinct()`	tekrarları atar (`equals`)
`limit(n)` / `skip(n)`	ilk *n* öğeyi alır / atlar
`peek(f)`	öğeleri izler (hata ayıklama)

Uç işlem	Sonuç
`collect(...)`	liste, küme, eşlem…
`forEach(f)`	her öğe için bir eylem
`count()`	öğe sayısı (`long`)
`reduce(...)`	öğeleri tek değere indirger
`findFirst()`	ilk öğe (`Optional`)
`anyMatch` / `allMatch` / `noneMatch`	`boolean`
`min` / `max`	`Optional`

Ara işlem **yeni bir akış** döndürür ve zincir sürer; uç işlem akış dışında bir sonuç (veya hiçbir şey) döndürür ve hattı **çalıştırır**. Uç işlemi olmayan bir hat hiçbir şey yapmaz.

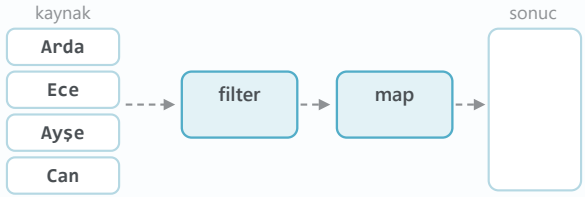
Tembel Değerlendirme ve Tek Kullanım

```
List<String> adlar =
    Arrays.asList("Arda", "Ece", "Ayşe", "Can");
Stream<String> hat = adlar.stream()
    .filter(s -> s.startsWith("A"))
    .map(String::toUpperCase);
// Henüz hiçbir öğe işlenmedi.
List<String> sonuc =
    hat.collect(Collectors.toList());
System.out.println(sonuc);  // [ARDA, AYŞE]
hat.count();   // IllegalStateException
```

- **Tembel** (lazy): ara işlemler yalnızca bir **tarif** kaydeder; öğeler uç işlem çağrılınca işlenir.
- Öğeler hattan **tek tek** geçer: ilk öğe `filter` ve `map` 'ten geçer, sonra ikincisi gelir.
- `findFirst`, `limit` sonucu bulunca durur; kalan öğeler hiç işlenmez.
- Akış **tek kullanımlıktır**: uç işlemden sonra yeniden kullanmak `IllegalStateException` fırlatır; gerekirse kaynaktan yeni akış alınır.

Tembel Akış Hattı: Adım Adım

```
▶ List<String> adlar = Arrays.asList("Arda", "Ece", "Ayşe",
  "Can");
▶ Stream<String> hat = adlar.stream()
▶     .filter(s -> { System.out.println("filter " + s);
▶                    return s.startsWith("A"); })
▶     .map(s -> { System.out.println("map " + s);
▶                 return s.toUpperCase(); });
7 System.out.println("hat hazır");
8 List<String> sonuc = hat.collect(Collectors.toList());
9 System.out.println(sonuc);
```



Hat kuruldu: `filter` ve `map` yalnızca kaydedildi. Hiçbir lambda çalışmadı, çıktı boş.

Adım 1 / 10

DEĞİŞKENLER

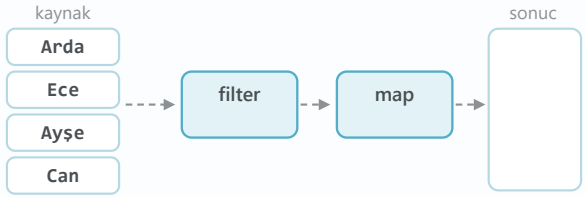
`adlar = [Arda, Ece, Ayşe, Can]`

`hat = Stream`

ÇIKTI

Tembel Akış Hattı: Adım Adım

```
1 List<String> adlar = Arrays.asList("Arda", "Ece", "Ayşe",
  "Can");
2 Stream<String> hat = adlar.stream()
3     .filter(s -> { System.out.println("filter " + s);
4                    return s.startsWith("A"); })
5     .map(s -> { System.out.println("map " + s);
6                 return s.toUpperCase(); });
▶ System.out.println("hat hazır");
8 List<String> sonuc = hat.collect(Collectors.toList());
9 System.out.println(sonuc);
```



Bu satır, işlemlerden **önce** yazılır: ara işlemler tembeldir.

Adım 2 / 10

DEĞİŞKENLER

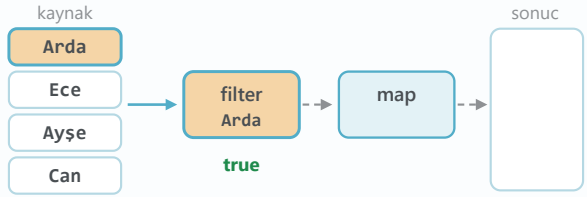
`hat = Stream`

ÇIKTI

```
hat hazır
```

Tembel Akış Hattı: Adım Adım

```
1 List<String> adlar = Arrays.asList("Arda", "Ece", "Ayşe",
  "Can");
2 Stream<String> hat = adlar.stream()
▶     .filter(s -> { System.out.println("filter " + s);
▶                    return s.startsWith("A"); })
5     .map(s -> { System.out.println("map " + s);
6                 return s.toUpperCase(); });
7 System.out.println("hat hazır");
▶ List<String> sonuc = hat.collect(Collectors.toList());
9 System.out.println(sonuc);
```



`collect` hattı çalıştırır. İlk öğe `Arda` `filter`'a girer: "A" ile başlıyor → geçer.

Adım 3 / 10

DEĞİŞKENLER

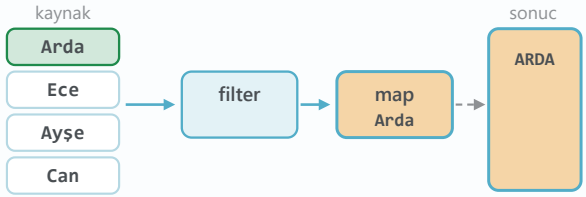
`s = "Arda"`

ÇIKTI

```
hat hazır
filter Arda
```

Tembel Akış Hattı: Adım Adım

```
1  List<String> adlar = Arrays.asList("Arda", "Ece", "Ayşe",
   "Can");
2  Stream<String> hat = adlar.stream()
3      .filter(s -> { System.out.println("filter " + s);
4                     return s.startsWith("A"); })
▶      .map(s -> { System.out.println("map " + s);
▶                  return s.toUpperCase(); });
7  System.out.println("hat hazır");
▶  List<String> sonuc = hat.collect(Collectors.toList());
9  System.out.println(sonuc);
```



Aynı öğe hemen `map`'e geçer ve `ARDA` sonuca eklenir. İkinci öğeye ancak şimdi sıra gelir.

Adım 4 / 10

DEĞİŞKENLER

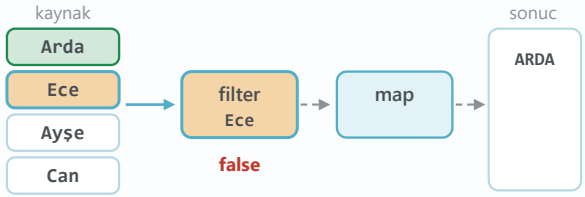
`s = "Arda"`

ÇIKTI

```
hat hazır
filter Arda
map Arda
```

Tembel Akış Hattı: Adım Adım

```
1  List<String> adlar = Arrays.asList("Arda", "Ece", "Ayşe",
   "Can");
2  Stream<String> hat = adlar.stream()
       .filter(s -> { System.out.println("filter " + s);
                      return s.startsWith("A"); })
5      .map(s -> { System.out.println("map " + s);
                   return s.toUpperCase(); });
7  System.out.println("hat hazır");
   List<String> sonuc = hat.collect(Collectors.toList());
9  System.out.println(sonuc);
```



`Ece` koşulu sağlamaz: elenir, `map` 'e hiç ulaşmaz.

Adım 5 / 10

DEĞİŞKENLER

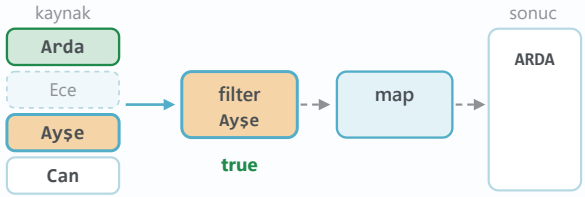
`s = "Ece"`

ÇIKTI

```
hat hazır
filter Arda
map Arda
filter Ece
```

Tembel Akış Hattı: Adım Adım

```
1 List<String> adlar = Arrays.asList("Arda", "Ece", "Ayşe",
  "Can");
2 Stream<String> hat = adlar.stream()
▶     .filter(s -> { System.out.println("filter " + s);
▶                    return s.startsWith("A"); })
5     .map(s -> { System.out.println("map " + s);
6                 return s.toUpperCase(); });
7 System.out.println("hat hazır");
▶ List<String> sonuc = hat.collect(Collectors.toList());
9 System.out.println(sonuc);
```



`Ayşe` filtreyi geçer.

Adım 6 / 10

DEĞİŞKENLER

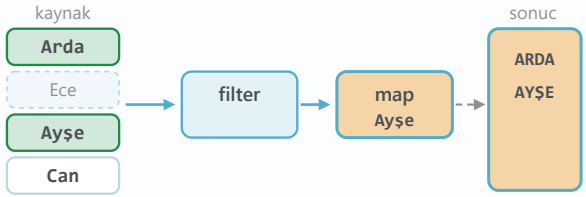
`s = "Ayşe"`

ÇIKTI

```
hat hazır
filter Arda
map Arda
filter Ece
filter Ayşe
```

Tembel Akış Hattı: Adım Adım

```
1  List<String> adlar = Arrays.asList("Arda", "Ece", "Ayşe",
   "Can");
2  Stream<String> hat = adlar.stream()
3      .filter(s -> { System.out.println("filter " + s);
4                     return s.startsWith("A"); })
▶      .map(s -> { System.out.println("map " + s);
▶                  return s.toUpperCase(); });
7  System.out.println("hat hazır");
▶  List<String> sonuc = hat.collect(Collectors.toList());
9  System.out.println(sonuc);
```



`map` büyük harfe çevirir: `AYŞE` sonuca eklenir.

Adım 7 / 10

DEĞİŞKENLER

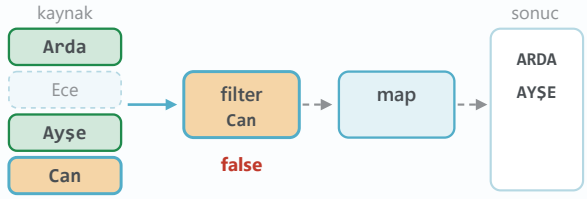
`s = "Ayşe"`

ÇIKTI

```
hat hazır
filter Arda
map Arda
filter Ece
filter Ayşe
map Ayşe
```

Tembel Akış Hattı: Adım Adım

```
1  List<String> adlar = Arrays.asList("Arda", "Ece", "Ayşe",
   "Can");
2  Stream<String> hat = adlar.stream()
       .filter(s -> { System.out.println("filter " + s);
                      return s.startsWith("A"); })
5      .map(s -> { System.out.println("map " + s);
                   return s.toUpperCase(); });
7  System.out.println("hat hazır");
   List<String> sonuc = hat.collect(Collectors.toList());
9  System.out.println(sonuc);
```



`Can` elenir. Kaynakta öğe kalmadı.

Adım 8 / 10

DEĞİŞKENLER

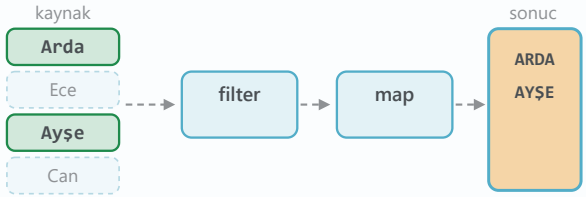
`s = "Can"`

ÇIKTI

```
hat hazır
filter Arda
map Arda
filter Ece
filter Ayşe
map Ayşe
filter Can
```

Tembel Akış Hattı: Adım Adım

```
1 List<String> adlar = Arrays.asList("Arda", "Ece", "Ayşe",
  "Can");
2 Stream<String> hat = adlar.stream()
3     .filter(s -> { System.out.println("filter " + s);
4                    return s.startsWith("A"); })
5     .map(s -> { System.out.println("map " + s);
6                 return s.toUpperCase(); });
7 System.out.println("hat hazır");
▶ List<String> sonuc = hat.collect(Collectors.toList());
9 System.out.println(sonuc);
```



`collect` listeyi döndürür. `filter` 4 kez, `map` yalnızca 2 kez çalıştı.

Adım 9 / 10

DEĞİŞKENLER

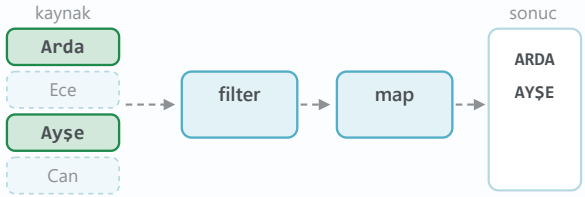
`sonuc = [ARDA, AYŞE]`

ÇIKTI

```
hat hazır
filter Arda
map Arda
filter Ece
filter Ayşe
map Ayşe
filter Can
```

Tembel Akış Hattı: Adım Adım

```
1 List<String> adlar = Arrays.asList("Arda", "Ece", "Ayşe",
  "Can");
2 Stream<String> hat = adlar.stream()
3     .filter(s -> { System.out.println("filter " + s);
4                    return s.startsWith("A"); })
5     .map(s -> { System.out.println("map " + s);
6                 return s.toUpperCase(); });
7 System.out.println("hat hazır");
8 List<String> sonuc = hat.collect(Collectors.toList());
▶ System.out.println(sonuc);
```



Sonuç yazılır. Çıktının sırası, öğelerin hattan **tek tek** geçtiğini gösterir.

Adım 10 / 10

DEĞİŞKENLER

`sonuc = [ARDA, AYŞE]`

ÇIKTI

```
hat hazır
filter Arda
map Arda
filter Ece
filter Ayşe
map Ayşe
filter Can
[ARDA, AYŞE]
```

`reduce`, `Collectors` ve İlkel Akışlar

```
List<String> meyveler = Arrays.asList(
    "elma", "kivi", "muz", "erik", "incir");
int toplam = Arrays.asList(4, 8, 15, 16, 23)
    .stream().reduce(0, Integer::sum);
Map<Integer, List<String>> boy = meyveler
    .stream().collect(
        Collectors.groupingBy(String::length));
String hepsi = meyveler.stream()
    .collect(Collectors.joining(", "));
int t = IntStream.rangeClosed(1, 100).sum();
Optional<String> z = meyveler.stream()
    .filter(m -> m.startsWith("z")).findFirst();
System.out.println(z.orElse("yok"));
```

DEĞERLER

```
toplam = 66
boy = {3=[muz],
  4=[elma, kivi, erik],
  5=[incir]}
hepsi = elma, kivi, muz,
  erik, incir
t = 5050
yok
```

- `reduce` : 0 + 4 + 8 + 15 + 16 + 23.
- `groupingBy` gruplar, `joining` birleştirir.
- `IntStream` : kutulamasız sayı akışı; `sum` , `average` hazır.
- `Optional` : sonuç olmayabilir; `orElse` , `ifPresent` .

AKIŞ API'Sİ

Yan Etkiler ve Paralel Akışlar

```
List<Integer> liste = new ArrayList<>();
IntStream.range(0, 10000).parallel()
    .forEach(liste::add);          // YANLIŞ: paylaşılan liste

List<Integer> dogru = IntStream.range(0, 10000).parallel()
    .boxed()
    .collect(Collectors.toList()); // DOĞRU: 10000 öğe
```

BİRİNCİ HATTIN SONUCU (HER ÇALIŞTIRMADA FARKLI)

```
ArrayIndexOutOfBoundsException
ya da 10000'den az öğe
```

- Lambdalar dış değişkenleri **değiştirmemeli**; sonuç `map`, `collect`, `reduce` ile üretilmeli.
- `parallel()` / `parallelStream()` işi ortak `ForkJoinPool` iş parçacıklarına böler. `ArrayList` iş parçacığı güvenli olmadığı için ilk hat bozulur.
- Paralel akış **her zaman hızlı değildir**: küçük veri, ucuz işlemler, G/Ç bekleyen işler ve sıraya bağlı işlemler (`sorted`, `findFirst`) yavaşlatabilir; önce ölçülür.

Sık hatalar: akışı ikinci kez kullanmak; asıl işi `peek` içinde yapmak; `forEach` ile dış koleksiyon doldurmak; paralel akışın kendiliğinden hızlı olduğunu varsaymak.

SEÇİM

Hangi Koleksiyonu Seçmeli?

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

Değişken arayüz tipinde tanımlanınca (`List` , `Set` , `Map` , `Queue` , `Deque`) ihtiyaç değiştiğinde 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` öğe kabul edilmez; `Set.of(1, 1)` → `IllegalArgumentException`.

SALT OKUNUR GÖRÜNÜM (JAVA 1.2+)

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

Gezinme: `for-each` veya `Iterator` (silme için). Algoritmalar: `Collections.sort`, `shuffle`, `binarySearch`, `copy`, `fill`, `reverse`, `swap`. Akışlar: `filter`, `map`, `sorted` (ara, tembel) ve `collect`, `reduce`, `count` (uç).

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.

Akış Boru Hattı

Simülatör: `filter`, `map`, `sorted`, `limit` aşamalarından öğelerin tek tek geçişi, tembel değerlendirme.

Koleksiyonlarla çalışırken önce doğru arayüz, sonra gerçekleme seçilir. Jenerik tipler hataları derleme zamanında yakalar.

PRATİK

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: doğru tip `List<Integer>`'dır. `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. Sayı dizgi olarak saklanacaksa `String.valueOf(42)` eklenir.

PRATİK

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(…)` verilir.

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 karşılaştırması `equals` ile yapılır.

PRATİK

Sık Yapılan Hatalar (3/4)

ARRAYS.ASLİST 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 gezilir.

PRATİK

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 sırasını `Collator` (java.text) verir.

PRATİK

Örnek Problemler (1)

```java
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 öğe vardır? Son üç öğe nedir?
3. `fibonacci(1)` çağrısında döngü kaç kez döner?
4. Bu soru için liste gerçekten gerekli mi?

▶ **Cevapları göster**

PRATİK

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**

PRATİK

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**

PRATİK

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 çıktısı nedir? (`push` = `offerFirst`, `pop` = `removeFirst`.)

▶ **Cevapları göster**

PRATİK

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**

PRATİK

Alıştırmalar (2)

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

▶ **Cevapları göster**

PRATİK

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**

ÖZET

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 6: İstisnalar · Terim sözlüğü