

# JAVA STREAM API

## FONKSİYONEL PROGRAMLAMA

Filter, Map, Reduce, Collect, Parallel Streams

# GİRİŞ: STREAM API NEDİR?

## Stream Nedir?

**Stream = Veri akışı, koleksiyonlar üzerinde fonksiyonel işlemler**

Stream ≠ Collection:

- **Collection:** Veriyi depolar
- **Stream:** Veriyi işler (depolamaz)

Neden Stream?

- Daha okunabilir kod
- Daha az kod
- Fonksiyonel programlama
- Paralel işleme desteği

Eski Yol vs Stream:

```
import java.util.*;

public class StreamGiris {
    public static void main(String[] args) {

        List<Integer> sayilar = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9,
10);

        // ESKİ YOL - Çift sayıların toplamı
        int toplam1 = 0;
        for (Integer sayi : sayilar) {
            if (sayi % 2 == 0) {
                toplam1 += sayi;
            }
        }
        System.out.println("Eski yol: " + toplam1);

        // YENİ YOL - Stream API
        int toplam2 = sayilar.stream()
            .filter(sayi -> sayi % 2 == 0)
            .mapToInt(Integer::intValue)
            .sum();
        System.out.println("Stream: " + toplam2);
    }
}
```

Çıktı:

```
Eski yol: 30
Stream: 30
```

# 1. STREAM OLUŞTURMA

## 1.1 Farklı Kaynaklardan Stream

```
import java.util.*;
import java.util.stream.*;

public class StreamOlusturma {
    public static void main(String[] args) {

        // 1. COLLECTION'DAN
        List<String> liste = Arrays.asList("A", "B", "C");
        Stream<String> stream1 = liste.stream();

        // 2. DİZİDEN
        String[] dizi = {"X", "Y", "Z"};
        Stream<String> stream2 = Arrays.stream(dizi);

        // 3. Stream.of() ile
        Stream<Integer> stream3 = Stream.of(1, 2, 3, 4, 5);

        // 4. Stream.iterate() ile - Sonsuz stream
        Stream<Integer> stream4 = Stream.iterate(0, n -> n + 2)
            .limit(5);
        stream4.forEach(n -> System.out.print(n + " "));
        // Çıktı: 0 2 4 6 8

        // 5. Stream.generate() ile
        Stream<Double> stream5 = Stream.generate(Math::random)
            .limit(3);

        // 6. IntStream, LongStream, DoubleStream
        IntStream intStream = IntStream.range(1, 6); // 1,2,3,4,5
        IntStream intStream2 = IntStream.rangeClosed(1, 5);

        // 7. BOŞ STREAM
        Stream<String> bosStream = Stream.empty();
    }
}
```

## 2. INTERMEDIATE OPERATIONS

### 2.1 filter() - Filtreleme

```
import java.util.*;

public class FilterOrnek {
    public static void main(String[] args) {

        List<Integer> sayilar = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9,
10);

        // Çift sayıları filtrele
        System.out.println("=== ÇİFT SAYILAR ===");
        sayilar.stream()
            .filter(n -> n % 2 == 0)
            .forEach(n -> System.out.print(n + " "));

        // 5'ten büyük sayılar
        System.out.println("\n\n=== 5'TEN BÜYÜK ===");
        sayilar.stream()
            .filter(n -> n > 5)
            .forEach(n -> System.out.print(n + " "));

        // ÇOKLU FİLTRE
        System.out.println("\n\n=== ÇİFT VE 5'TEN BÜYÜK ===");
        sayilar.stream()
            .filter(n -> n % 2 == 0)
            .filter(n -> n > 5)
            .forEach(n -> System.out.print(n + " "));
    }
}
```

### 2.2 map() - Dönüştürme

```
import java.util.*;

public class MapOrnek {
    public static void main(String[] args) {

        List<Integer> sayilar = Arrays.asList(1, 2, 3, 4, 5);

        // Sayıların karesi
        System.out.println("=== KARELER ===");
        sayilar.stream()
            .map(n -> n * n)
            .forEach(n -> System.out.print(n + " "));

        // STRING DÖNÜŞÜM
        List<String> isimler = Arrays.asList("ali", "ayşe", "mehmet");
```

```

        System.out.println("\n\n=== BÜYÜK HARF ===");
        isimler.stream()
            .map(String::toUpperCase)
            .forEach(System.out::println);

        System.out.println("\n=== UZUNLUKLAR ===");
        isimler.stream()
            .map(String::length)
            .forEach(n -> System.out.print(n + " "));
    }
}

```

### 2.3 flatMap() - Düzleştirme

```

import java.util.*;

public class FlatMapOrnek {
    public static void main(String[] args) {

        // İç içe listeler
        List<List<Integer>> listeler = Arrays.asList(
            Arrays.asList(1, 2, 3),
            Arrays.asList(4, 5, 6),
            Arrays.asList(7, 8, 9)
        );

        // flatMap() ile düzleştir
        System.out.println("=== FLATMAP ===");
        listeler.stream()
            .flatMap(liste -> liste.stream())
            .forEach(n -> System.out.print(n + " "));

        // STRING ÖRNEK
        List<String> cumleler = Arrays.asList(
            "Merhaba Dünya",
            "Java Stream API"
        );

        // Her cümledeki kelimeleri ayrı ayrı yazdır
        System.out.println("\n\n=== TÜM KELİMELER ===");
        cumleler.stream()
            .flatMap(cumle -> Arrays.stream(cumle.split(" ")))
            .forEach(System.out::println);
    }
}

```

### 2.4 distinct() - Benzersiz Elemanlar

```

import java.util.*;

public class DistinctOrnek {

```

```

public static void main(String[] args) {

    List<Integer> sayilar = Arrays.asList(1, 2, 2, 3, 3, 3, 4, 4, 5);

    System.out.println("=== ORJİNAL ===");
    sayilar.forEach(n -> System.out.print(n + " "));

    System.out.println("\n\n=== BENZERSİZ ===");
    sayilar.stream()
        .distinct()
        .forEach(n -> System.out.print(n + " "));

    // BENZERSİZ + SIRALAMA
    System.out.println("\n\n=== BENZERSİZ + SIRALI ===");
    sayilar.stream()
        .distinct()
        .sorted()
        .forEach(n -> System.out.print(n + " "));

}
}

```

## 2.5 sorted() - Sıralama

```

import java.util.*;

public class SortedOrnek {
    public static void main(String[] args) {

        List<Integer> sayilar = Arrays.asList(5, 2, 8, 1, 9, 3);

        // KÜÇÜKTEN BÜYÜĞE
        System.out.println("=== KÜÇÜKTEN BÜYÜĞE ===");
        sayilar.stream()
            .sorted()
            .forEach(n -> System.out.print(n + " "));

        // BÜYÜKTEN KÜÇÜĞE
        System.out.println("\n\n=== BÜYÜKTEN KÜÇÜĞE ===");
        sayilar.stream()
            .sorted(Comparator.reverseOrder())
            .forEach(n -> System.out.print(n + " "));

        // STRING SIRALAMA
        List<String> isimler = Arrays.asList("Mehmet", "Ali", "Zeynep");

        System.out.println("\n\n=== ALFABETİK ===");
        isimler.stream()
            .sorted()
            .forEach(System.out::println);

        // UZUNLUĞA GÖRE
        System.out.println("\n\n=== UZUNLUĞA GÖRE ===");
    }
}

```

```

        isimler.stream()
            .sorted(Comparator.comparing(String::length))
            .forEach(System.out::println);
    }
}

```

## 2.6 limit() ve skip()

```

import java.util.*;

public class LimitSkipOrnek {
    public static void main(String[] args) {

        List<Integer> sayilar = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9,
10);

        // İLK 5 ELEMAN
        System.out.println("=== İLK 5 ===");
        sayilar.stream()
            .limit(5)
            .forEach(n -> System.out.print(n + " "));

        // İLK 3'Ü ATLA
        System.out.println("\n\n=== İLK 3 ATLA ===");
        sayilar.stream()
            .skip(3)
            .forEach(n -> System.out.print(n + " "));

        // SKIP + LİMİT
        System.out.println("\n\n=== SKIP + LİMİT ===");
        sayilar.stream()
            .skip(3)
            .limit(5)
            .forEach(n -> System.out.print(n + " "));

        // SAYFALAMA
        System.out.println("\n\n=== SAYFALAMA ===");
        int sayfaBoyutu = 3;
        int sayfaNo = 2;

        sayilar.stream()
            .skip(sayfaNo * sayfaBoyutu)
            .limit(sayfaBoyutu)
            .forEach(n -> System.out.print(n + " "));
    }
}

```

## 3. TERMINAL OPERATIONS

### 3.1 collect() - Toplama

```
import java.util.*;
import java.util.stream.*;

public class CollectOrnek {
    public static void main(String[] args) {

        List<String> isimler = Arrays.asList("Ali", "Ayşe", "Mehmet");

        // LIST'E TOPLAMA
        List<String> liste = isimler.stream()
                                    .filter(s -> s.length() > 3)
                                    .collect(Collectors.toList());
        System.out.println("Liste: " + liste);

        // SET'E TOPLAMA
        Set<String> set = isimler.stream()
                                 .collect(Collectors.toSet());
        System.out.println("Set: " + set);

        // MAP'E TOPLAMA
        Map<String, Integer> map = isimler.stream()
                                           .collect(Collectors.toMap(
                                               isim -> isim,
                                               isim -> isim.length()
                                           ));
        System.out.println("Map: " + map);

        // BİRLEŞTİRME
        String birlesik = isimler.stream()
                                  .collect(Collectors.joining(", "));
        System.out.println("Birleşik: " + birlesik);
    }
}
```

### 3.2 count(), min(), max(), sum(), average()

```
import java.util.*;

public class AggregateOrnek {
    public static void main(String[] args) {

        List<Integer> sayilar = Arrays.asList(5, 2, 8, 1, 9, 3, 7);

        // COUNT
        long adet = sayilar.stream()
                           .filter(n -> n > 5)
```

```

        .count();
System.out.println("5'ten büyük: " + adet);

// MIN
Optional<Integer> min = sayilar.stream()
    .min(Integer::compareTo);
min.ifPresent(m -> System.out.println("Min: " + m));

// MAX
Optional<Integer> max = sayilar.stream()
    .max(Integer::compareTo);
max.ifPresent(m -> System.out.println("Max: " + m));

// SUM
int toplam = sayilar.stream()
    .mapToInt(Integer::intValue)
    .sum();
System.out.println("Toplam: " + toplam);

// AVERAGE
OptionalDouble ortalama = sayilar.stream()
    .mapToInt(Integer::intValue)
    .average();

ortalama.ifPresent(ort ->
    System.out.println("Ortalama: " + ort));
    }
}

```

### 3.3 anyMatch(), allMatch(), noneMatch()

```

import java.util.*;

public class MatchOrnek {
    public static void main(String[] args) {

        List<Integer> sayilar = Arrays.asList(2, 4, 6, 8, 10);

        // anyMatch - Herhangi biri uyuyor mu?
        boolean varMi5 = sayilar.stream()
            .anyMatch(n -> n > 5);
        System.out.println("5'ten büyük var mı? " + varMi5);

        // allMatch - Hepsi uyuyor mu?
        boolean hepsiCift = sayilar.stream()
            .allMatch(n -> n % 2 == 0);
        System.out.println("Hepsi çift mi? " + hepsiCift);

        // noneMatch - Hiçbiri uymuyor mu?
        boolean hicTek = sayilar.stream()
            .noneMatch(n -> n % 2 == 1);
        System.out.println("Hiç tek yok mu? " + hicTek);
    }
}

```

```
}
```

### 3.4 reduce() - İndirgeme

```
import java.util.*;

public class ReduceOrnek {
    public static void main(String[] args) {

        List<Integer> sayilar = Arrays.asList(1, 2, 3, 4, 5);

        // TOPLAMA
        Optional<Integer> toplam = sayilar.stream()
            .reduce((a, b) -> a + b);
        toplam.ifPresent(t -> System.out.println("Toplam: " + t));

        // Başlangıç değeri ile
        int toplam2 = sayilar.stream()
            .reduce(0, (a, b) -> a + b);
        System.out.println("Toplam2: " + toplam2);

        // ÇARPMA
        int carpim = sayilar.stream()
            .reduce(1, (a, b) -> a * b);
        System.out.println("Çarpım: " + carpim);

        // MAKSİMUM
        Optional<Integer> max = sayilar.stream()
            .reduce((a, b) -> a > b ? a : b);
        max.ifPresent(m -> System.out.println("Max: " + m));
    }
}
```

## 4. PARALLEL STREAMS

### 4.1 Parallel Stream Nedir?

**Parallel Stream = Stream işlemlerini çoklu çekirdeklerde paralel çalıştırma**

Ne zaman kullanılır?

- Büyük veri setleri
- CPU-yoğun işlemler
- Bağımsız işlemler

### 4.2 Sequential vs Parallel

```
import java.util.*;
import java.util.stream.*;

public class ParallelPerformans {
    public static void main(String[] args) {

        List<Integer> sayilar = IntStream.rangeClosed(1, 10_000_000)
                                          .boxed()
                                          .collect(Collectors.toList());

        // SEQUENTIAL STREAM
        long baslangic1 = System.currentTimeMillis();
        long toplam1 = sayilar.stream()
                              .mapToLong(Integer::longValue)
                              .sum();
        long bitis1 = System.currentTimeMillis();

        System.out.println("=== SEQUENTIAL ===");
        System.out.println("Toplam: " + toplam1);
        System.out.println("Süre: " + (bitis1 - baslangic1) + " ms");

        // PARALLEL STREAM
        long baslangic2 = System.currentTimeMillis();
        long toplam2 = sayilar.parallelStream()
                              .mapToLong(Integer::longValue)
                              .sum();
        long bitis2 = System.currentTimeMillis();

        System.out.println("\n=== PARALLEL ===");
        System.out.println("Toplam: " + toplam2);
        System.out.println("Süre: " + (bitis2 - baslangic2) + " ms");

        // HIZLANMA
        double hizlanma = (double)(bitis1 - baslangic1) /
                          (bitis2 - baslangic2);
        System.out.printf("\nHızlanma: %.2fx\n", hizlanma);
    }
}
```

### 4.3 Parallel Stream Kullanımı

```
import java.util.*;

public class ParallelKullanim {
    public static void main(String[] args) {

        List<Integer> sayilar = Arrays.asList(1, 2, 3, 4, 5);

        // 1. parallelStream() ile
        sayilar.parallelStream()
            .forEach(n -> System.out.println(
                Thread.currentThread().getName() + ": " + n));

        // 2. parallel() ile
        sayilar.stream()
            .parallel()
            .forEach(n -> System.out.println(n));

        // 3. Paralel durumu kontrol
        boolean paralel1 = sayilar.stream().isParallel();
        boolean paralel2 = sayilar.parallelStream().isParallel();

        System.out.println("stream(): " + paralel1);
        System.out.println("parallelStream(): " + paralel2);
    }
}
```

### 4.4 Dikkat Edilecekler

```
import java.util.*;
import java.util.stream.*;

public class ParallelDikkat {

    // YANLIŞ - Thread-safe değil!
    public static void yanlisKullanim() {
        List<Integer> sonuc = new ArrayList<>();

        IntStream.range(1, 1000)
            .parallel()
            .forEach(sonuc::add); // SORUNLU!

        // Sonuç değişken, 999'dan az olabilir!
    }

    // DOĞRU - collect() kullan
    public static void dogruKullanim() {
        List<Integer> sonuc = IntStream.range(1, 1000)
            .parallel()
            .boxed()
            .collect(Collectors.toList());
    }
}
```

```
        // Her zaman 999 eleman  
    }  
}
```

## 5. BEST PRACTICES

### 5.1 Yapılması Gerekenler

- Method reference kullan
- filter önce, map sonra
- Primitive stream kullan (IntStream, LongStream)
- Stateless lambda kullan

```
// ✅ DOĞRU: Method reference
sayilar.stream().forEach(System.out::println);

// ✅ DOĞRU: filter önce, map sonra
sayilar.stream()
    .filter(n -> n % 2 == 0)
    .map(n -> n * n)
    .count();

// ✅ DOĞRU: Primitive stream
int toplam = sayilar.stream()
    .mapToInt(Integer::intValue)
    .sum();
```

## SONUÇ

Bu dokümanda Java Stream API'nin tüm önemli konularını detaylı olarak inceledik:

- **Stream Oluşturma:** Collection, Array, iterate, generate
- **Intermediate Operations:** filter, map, flatMap, distinct, sorted, limit, skip
- **Terminal Operations:** collect, count, min, max, reduce, forEach
- **Parallel Streams:** Performans artışı ve dikkat edilecekler
- **Best Practices:** En iyi kullanım örnekleri

Stream API, Java'da fonksiyonel programlamanın temelidir. Koleksiyonlar üzerinde güçlü ve okunabilir işlemler yapmanızı sağlar.

### ÖNEMLİ HATIRLATMALAR

- Stream'ler tek kullanımlıktır
- filter önce, map sonra (performans)
- Parallel stream küçük listelerde yavaş olabilir
- Stateless lambda kullan
- Method reference daha temiz