

MODÜL-2
İLERİ JAVA VE
VERİTABANI

PAKET - 4
İLERİ JAVA



BÖLÜM-2 STREAM API

TECHPROED

STREAM API

- Büyük veri içeren nesnelerini **(Collection v.b)** **fonksiyonel programlama** ile işlememize imkan sağlayan bir API'dir.
- **Stream** bir veri yapısı değildir ve bellekte yer tutmaz. Sadece, var olan veri yapılarını girdi olarak alır ve girdilerin veri yapısını değiştirmeden işler.
- Verilerin sıralı işlemlerden (**pipeline**) geçirilerek işlenmesini ve istenilen sonuçların elde edilmesini sağlar.
- Stream API, ilk olarak **Java8** ile gelmiştir ve **java.util.stream** paketinde yer almaktadır.



TECHPROED

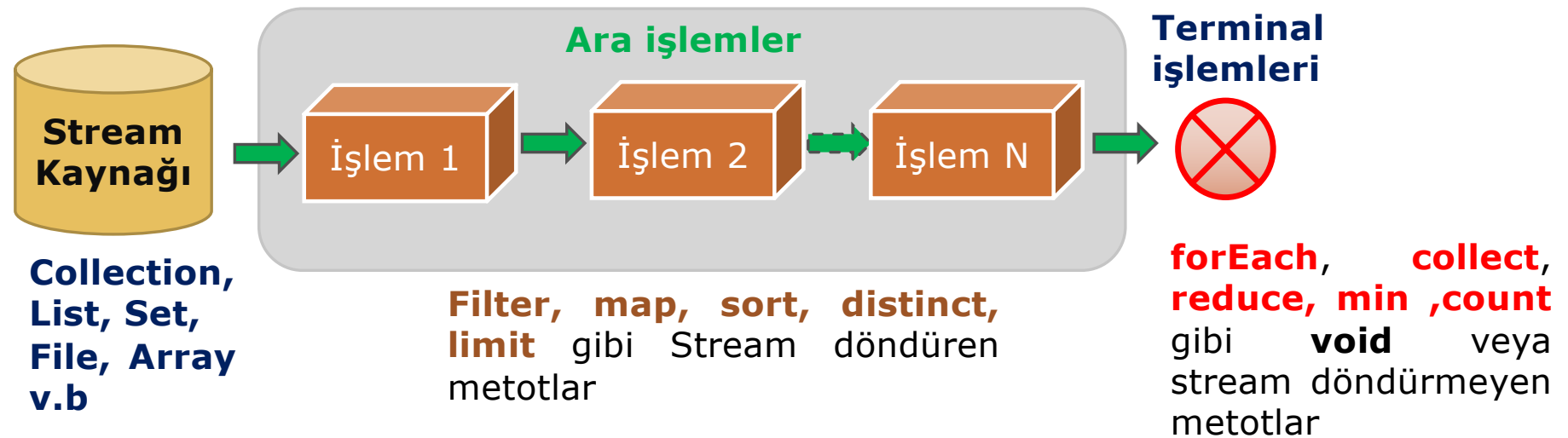
STREAM API

- **Stream API** içerisindeki metotlar **Lambda ifadelerini** desteklemektedir.
 - Metotlar içerisinde fonksiyonel arayüzler kullandığı için Lambda ifadelerini kullanmak mümkün.
- Döngü vb. işlemleri kullanmak yerine hazır metotların kullanımı sayesinde çok **daha kısa ve anlaşılır** kod yazmak mümkündür.
- **ParallelStreams** sayesinde **multi-threading** işlemleri yapmak daha kolaydır.
- Girdi olarak aldığı nesneleri (veri) değiştirmede için **daha güvenilir** program yazmak mümkündür.
- Dezavantajı ise geleneksel döngü tabanlı programlamaya göre çoğu durumda **daha yavaş sonuç** üretebilmektedir.

TECHPROED

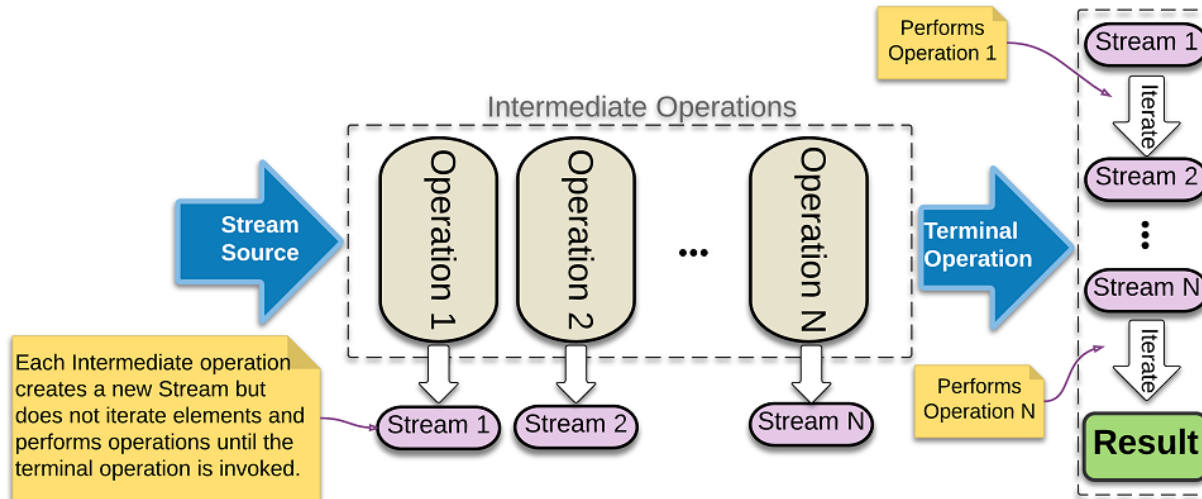
STREAM PIPELINE (HAT)

- Bir **Stream** hattı; bir **kaynak**, O'nu takip eden 0 veya daha fazla **ara işlem** ve bir **terminal** (sonlandırıcı) işlem içerir.



STREAM PIPELINE (HAT)

- Stream'lerde **ara işlemler**, **tembel (lazy)** olarak yürütülür. Yani **terminal** işlemi çağrılana kadar koşturulmazlar. Sadece, yeni bir **stream** nesnesi hazırlarlar.
- Terminal** işlemi çağrıldığında ise bu stream'ler alınarak **tek tek ara işlemler** gerçekleştirilir ve sonuç oluşturulur.

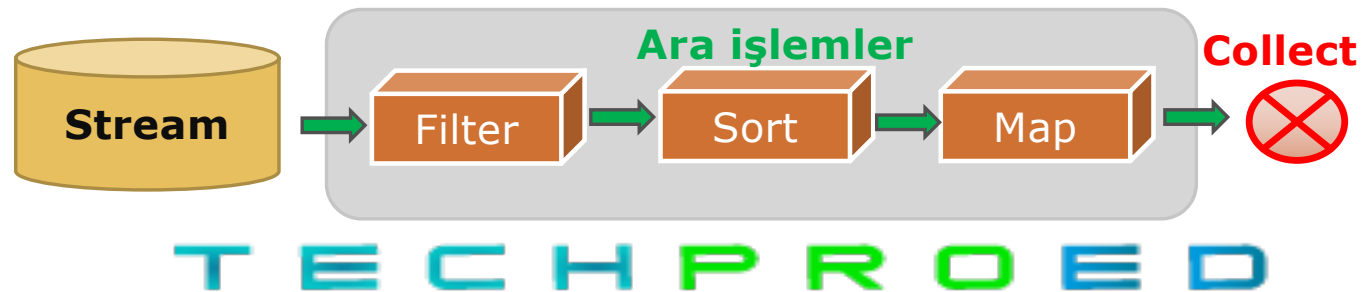


- Stream'lerin **büyük veri** gruplarında çalışacağı düşünüldüğünde terminal işlemini çağrılmadan tüm işlemleri baştan yapmak zaman kaybına yol açabilir.
- Belki de bazı işlemlerdeki veriler hiç kullanılmayabilir.
- Bu yüzden **tembel davranmak** daha efektif bir yöntemdir.

TECHPROED

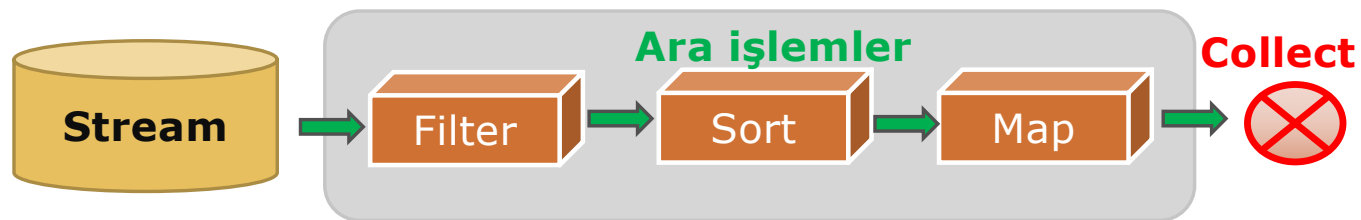
ARA İŞLEMLER

- Ara işlemler **0 yada daha fazla** olabilir.
- İşlemlerin **sırası** özellikle büyük miktardaki veriler için **önem arz eder**. Önce filtreleme (filter), sonra sıralama (sort) ve değiştirme (map).
- Büyük miktardaki veriler için **ParallelStream** kullanmak mantıklıdır.
- Yaygın kullanılan ara işlemlerden bazıları
 - **filter()** **map()** **sorted()** **distinct()** **limit()**
 - **findFirst()** **skip()** **flatMap()**



TERMINAL İŞLEMLERİ

- Terminal işlemi **Stream** nesnesini alır ve tüketir. (**eager**)
- Sadece tek terminal işlemi kullanılabilir.
- Yaygın kullanılan terminal işlemlerinden bazıları
 - **forEach()** **reduce()** **collect()**
 - **max()** **min()** **count()**



BİR STREAM NASIL OLUŞTURULUR?

- Her hangi **List**, **Set** gibi bir **Collection** **.stream()** metodu Stream'ler ile çalışabilir hale gelir.

```
List<Integer> liste = new ArrayList<>();  
liste.stream()
```

Artık pipeline'a girebilir.

- Herhangi bir dizi **Stream.of()** metodu yardımıyla Stream'ler ile çalışabilir hale gelir. Veya yeni bir Stream doğrudan oluşturulabilir.

```
Integer[] dizi = { 3, 1, 4, 1, 5, 9 };  
Stream<Integer> streamDizi =  
Stream.of(dizi);
```

```
Stream<String> kişiler = Stream.of("Ahmet", "Mahmet", "John");  
Stream<Integer> rakamlar = Stream.of(3, 1, 4, 1, 5, 9);
```

TECHPROED

ÖRNEK-1 (YAPISAL)

- Bir listeyi parametre olarak alan ve listedeki çift elemanları **YAN YANA** yazdıran metodu **yapısal** ve **fonksiyonel** olarak yazınız.

```
// A)"Yapısal programlama Kullanarak"  
private static void çiftElemanlarıYazdırYP(List<Integer> liste){  
    for(Integer w: liste){  
        if (w%2 == 0){  
            System.out.print(w + " ");  
        }  
    }  
}
```

TECHPROED

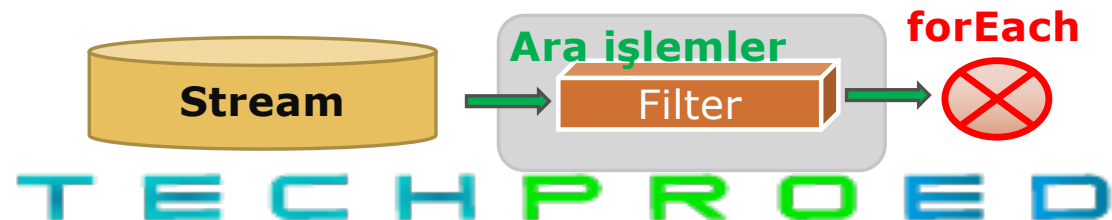
ÖRNEK-1 (FILTER VE FOREACH)

- Bir listeyi parametre olarak alan ve listedeki çift elemanları **YAN YANA** yazdıran metodu **yapısal** ve **fonksiyonel** olarak yazınız.

```
// B-"Fonksiyonel Programlama Kullanarak"  
//1.YÖNTEM: Lambda İfadeleri ile  
private static void çiftElemanlarıYazdırStream1(List<Integer> liste){  
    liste.stream().filter(t->t%2==0).forEach(t-> System.out.print(t + " "));  
}
```

Filter() : Streamdeki verileri içerisinde çağırılan fonksiyona göre filtreyen Ara işlem metodudur.

forEach(): Gelen verilerin tamamı işlenene (verilen metoda göre) veya bir exception oluşana kadar iterasyona devam eder. Terminal işlemdir. Stream'i kapatır.



ÖRNEK-1 (FILTER, FOREACH)

```
//2.YÖNTEM: Metot Referansı ve Java metotlarını kullanarak  
private static void çiftElemanlarıYazdırStream2(List<Integer> liste){  
    liste.stream().filter(t->t%2==0).forEach(System.out::print);  
    System.out.println();  
}
```

```
//3.YÖNTEM: Metot Referansı ve kendi metotumuz ile  
private static void çiftElemanlarıYazdırStream3(List<Integer> liste){  
    liste.stream().filter(t->t%2==0).forEach(Stream01::yazdır);  
}  
private static void yazdır(int a) {  
    System.out.print(a + " ");  
}
```

Stream01 ==> Yazdır Metotunun tanımlandığı class adı

TECHPROED

ÖRNEK-1

`List<Integer>`

10

1

8

5

`stream()`

10

1

8

5

`filter(t -> t%2==0)`

10



8



`forEach(print(t))`

10

8

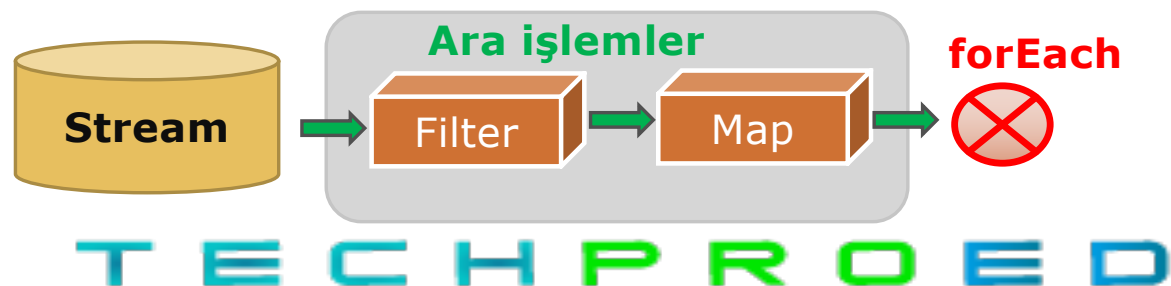
TECHPROED

ÖRNEK-2 (MAP)

- Bir listeyi parametre olarak alan ve listedeki her tek sayının karesini aralarında bir boşluk bırakarak konsola yazdıran metodu yazınız.

```
//1.YÖNTEM: Lambda ifadeleri kullanarak
private static void kareAl1(List<Integer> liste){
    liste.stream().filter(t->t%2!=0).
        map(t->t*t).
        forEach(t-> System.out.print(t + " "));
}
```

map() : Stream'deki verileri verilen metoda göre **değiştiren** (transformasyon) ara işlem metodudur.



ÖRNEK-2 (MAP)

- Bir listeyi parametre olarak alan ve listedeki her tek sayının karesini aralarında bir boşluk bırakarak konsola yazdıran metodu yazınız.

```
//2.YÖNTEM: Metot Referansı ve kendi metodumuzu kullanarak  
private static void kareAl2(List<Integer> list){  
    list.stream().filter(Stream01::tekMi).  
        map(Stream01::kareAl).  
        forEach(Stream01::yazdır);  
}  
  
private static boolean tekMi(int a){ return (a % 2 != 0);}   
private static int kareAl(int a){ return a*a;}
```

ÖRNEK-2

`List<Integer>`

10

1

8

5

`stream()`

10

1

8

5

`filter(t -> t%2!=0)`



1



5

`map(t -> t*t)`

1

25

`forEach(print(t))`

1

25

TECHPROED

ÖRNEK-3 (REDUCE)

- Bir listeyi parametre olarak alan ve listedeki tek sayıların **karelerinin toplamını** hesaplayan metodu yazınız.

```
private static Integer tekKareToplamı1(List<Integer> liste){  
    return liste.stream().  
        filter(t->t%2!=0).map(t->t*t).reduce(0, (x,y)->x+y);  
}
```

- İndirgeme (**reduction**) bir stream'ın **bir türe veya bir primitive'e** dönüştürülmesini sağlayan bir terminal işlemdir.
- Java 8 Stream API'de **average()**, **sum()**, **min()**, **max()** ve **count()** gibi tanımlanmış bir çok indirgeme metodu bulunmaktadır.
- Bu metotlar ilgili işlemleri gerçekleştirip **tek bir değer** döndürmektedir.
- reduce()** : Kendi indirgeme işlemlerimizi tanımlayabileceğimiz genel amaçlı bir metottur.

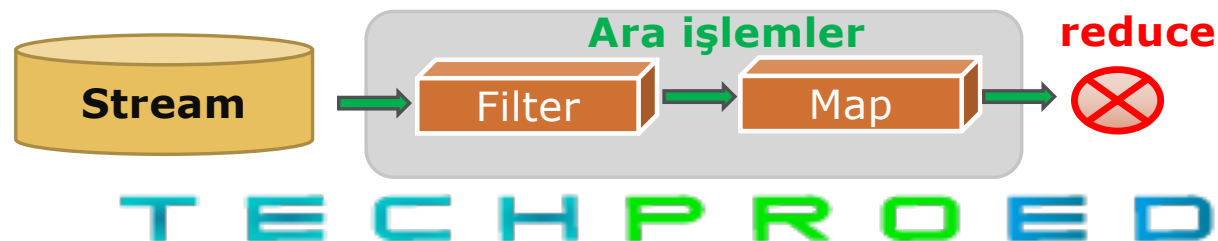
TECHPROED

ÖRNEK-3 (REDUCE)

- Bir listeyi parametre olarak alan ve listedeki tek sayıların **karelerinin toplamını** hesaplayan metodu yazınız.

*//2.YÖNTEM: Metot Referansı, Reduce için Java metodunun kullanımı
//Note: "Optional Class" kullanımı sayesinde null dönüş tipi ile uğraşmak daha kolaylaşıyor.*

```
private static Optional<Integer> tekKareToplamı2(List<Integer> liste){  
    return liste.stream().  
        filter(Stream01::tekMi).  
        map(Stream01::kareAl).  
        reduce(Integer::sum);    // reduce(Math::addExact);  
}
```



ÖRNEK - 3

`List<String>`

10

1

8

5

`.stream()`

10

1

8

5

`.filter(t-> t%2 !=0)`

1

5

`.map(t-> t*t)`

1

25

`.reduce(0, (x, y) -> x + y)`

26

25

TECHPROED

ÖRNEK-4 (COLLECT)

- Listedeki tek elemanların **karelerini** sıralayan ve **sonucu bir liste** olarak döndüren metodu yazınız. **NOT:** Tekrar eden elamanlar bir kere yazdırılmalı.
 - Tekrar eden elemanlar nasıl engellenebilir? **Distinct()**
 - Sıralama nasıl yapılır? **Sorted()**
 - Stream'in sonucu bir listeye nasıl saklanabilir? **Collect()**
- **collect()** Stream API'deki terminal işlemlerinden birisidir.
- Streamdeki elemanların çeşitli **veri yapılarına döndürülmesi** ve bazı ek işlemlerin uygulanmasını sağlamaktadır.
- **collect()** işlemindeki strateji **Collector** arayüzleri (interface) yardımıyla gerçekleştirilir.

TECHPROED

ÖRNEK-4 (COLLECT)

- Listedeki tek elemanların **karelerini** sıralayan ve **sonucu bir liste** olarak döndüren metodu yazınız. **NOT:** Tekrar eden elamanlar bir kere yazdırılmalı.

```
public static List<Integer> tekKareAlSıralı(List<Integer> list){  
    return list.stream().  
        filter(Stream01::tekMi).  
        distinct().  
        map(Stream01::kareAl).  
        sorted()      // sorted(Comparator.reverseOrder()).  
        collect(Collectors.toList());  
}
```



TECHPROED