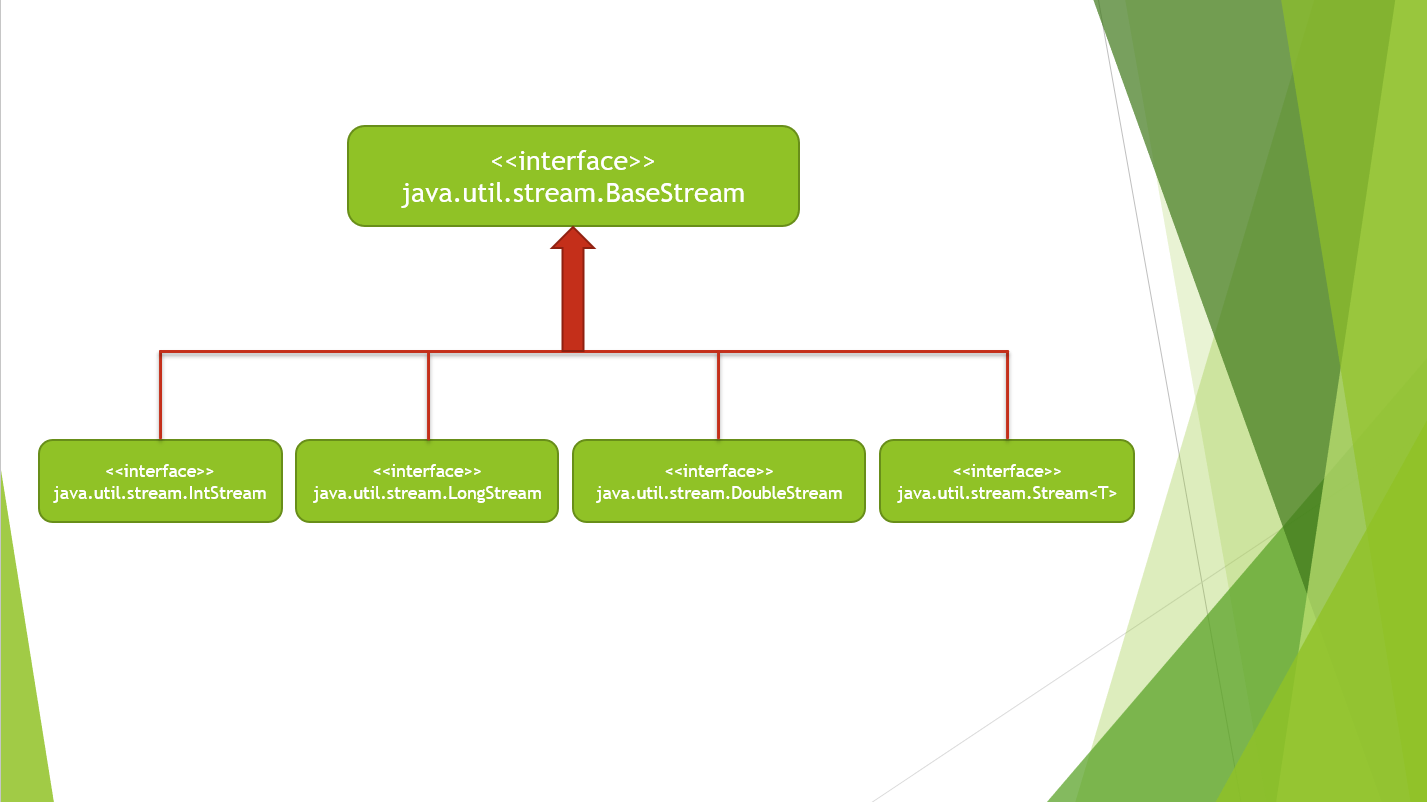
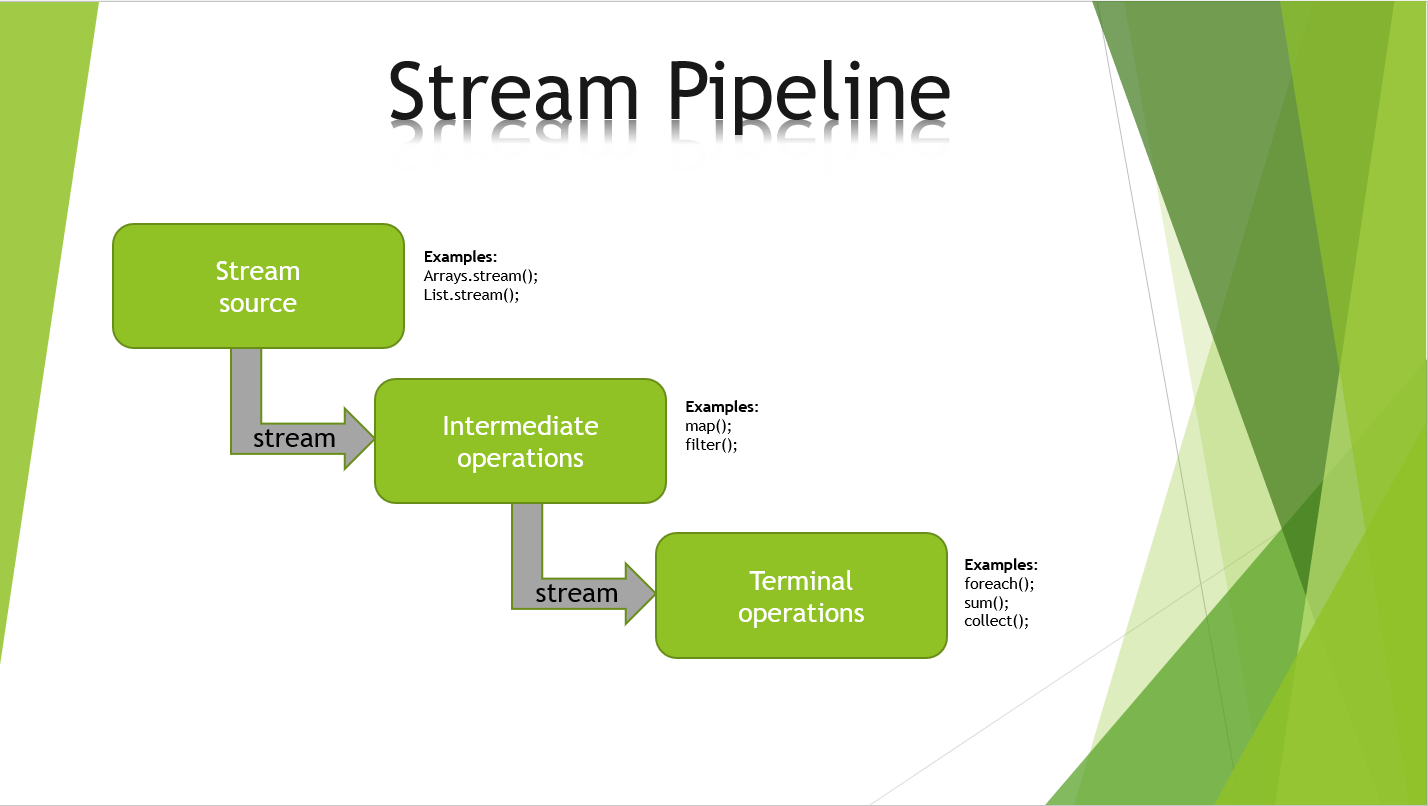
**STREAM**

Streamler BaseStrem arayüzünden türemektedir.

IntStream, DoubleStream, LongStream gibi primitive tipte olabileceği gibi Stream<T> Wrapper tipte de olabilir.



**STREAM PİPELİNE**



**Stream Source**

Kaynak datadır. Örnek oluşturulma şekilleri aşağıdadır.

Stream<String> empty = Stream.empty();

Stream<String> s1 = Stream.of("data1","data2","data3","data4");

List<String> list = Arrays.asList("a","b","c");

Stream<String> s2 = list.stream();

Stream<Double> s4 = Stream.generate(Math::random).limit(5);

Stream<Integer> s5 = Stream.iterate(1, n -> n+2).limit(5);

Stream<Integer> s3 = Arrays.stream(new Integer[] {1,2,3});

Stream<String> s6 = Pattern.compile(" ").splitAsStream("Java 8 stream test");

**Intermediate Operators**

Datanın işlenildiği dönüşümünün yapıldığı metotlardır.

Intermediate metotları geriye stream dönerler.

Lazy olarak çalışır. Terminal operator olmadan çalışmaz.

**limit()**

Stream API içerisinde yer alan limit operasyonu, Stream kaynağının belirli sayıda elemandan sonra sonlandırılmasını sağlamaktadır. Aksi taktirde Stream sonsuz döngüye girecektir. **generate**() veya **iterate**() gibi metotlardan sonra kullanılır.

**Filter**()

Filter source üzerinde eleme işlemi yapmaktadır. Verilen koşula uygun olmayan kaynak elemanları bir sonraki pipeline’ye dahil olmazlar. Böylelikle kaynak aslında bir süzgeçten geçirilmiş olur.

**Distinct()**

Distinct operatoru kaynağımız içerisindeki tekrar eden verileri silmektedir. Kelime anlamı itibariyle farklı demektir yani tekrar eden elemanlarımızı silerek tekrarları yok etmektedir. Set mantığı.

**Map()**

Stream içerisinde ki elemanları başka elamanlara dönüştürmek veya üzerlerinde işlem yapmak için kullanılmaktadır. mapToInt, mapToDouble, mapToLong gibi metotlar primitive Streamlere çevirmek için kullanılmaktadır.

**Peek()**

Stream Peek operasyonu Consumer alarak işlem gerçekleştirebilmektedir.

İntermedaiate operation olduğu için lazy olarak calisir. Terminal operation olmadan calismaz.

Bu yapının foreach’ten farkı peek’in non-terminate operasyon olmasıdır.

Peek kullanıldıktan sonra tekrardan başka Stream operasyonları kullanılabilirken foreach kullanıldıktan sonra Stream döngüsü sonlandığından dolayı bu mümkün olamamaktadır.

**Skip()**

Göz ardı edilecek verileri belirtir.

**Sorted**()

Veriler içerisinde sıralama yapmaktadır. Unicod değerine göre.

**flatMap() :**

Java 8 Stream içerisinde yer alan flatMap operasyonu birden fazla kaynağı birleştirip yeni bir Stream oluşturmaya yarar.

Aslında flatmap iki boyutlu olan kaynağımızı tek boyuta indirmek için kullanılıyor diyebiliriz. İçerisinde aynı tipte elemanlar olan iki farklı diziyi düşünelim bu dizileri birleştirmek için flatmap kullanabiliriz. Bu sayede iki kaynağımızda sanki tek bir kaynaktanmış gibi çalışacaktır.

**Generate()**

Supplier argüman alır ve supplier ın ürettiği veriden sonsuz bir stream üretir. Bu sonsuz stream supplierın ürettiği aynı veriyi sonsuz defa çoğaltıp stream içerisine koyar. Bu sonsuz stream biz bir limit vermediğimiz sürece memory dolana kadar stream objesini doldurmaya devam eder. (Exception in thread "main" java.lang.OutOfMemoryError: Java heap space) O nedenle üretilecek streamin boyutunu **limit**() metodu yardımıyla vermek gerekir.

**Terminal Operator**

Sonuç üreten metotlardır.

Stream üzerinde terminal operation çalıştıktan sonra, bir başka terminal operation’u çağıramayız.

**Count()**

Stream Count operasyonu Stream içerisinde yer alan toplam veri sayısının değerini verir, aslında list yapısından alışık olduğumuz size ve array yapısından alışık olduğumuz length değeri gibi düşünülebilir.

**FindAny() and FindFirst()**

FindAny paralel stream söz konusu olduğunda içerisinde yer alan herhangi bir elemanı döndürmektedir. Findany Stream operasyonu kullanıldıktan sonra direk olarak Stream kaynağımızdaki veri tipinde dönüş alamayız Optional adı verilen sınıflar kullanılır.

Findfirst operasyonunda ise Stream kaynağımız içerisinde yer alan ilk elemanı yine findanyde olduğu gibi Optional nesnesi olarak alabilmekteyiz.

**Foreach**()

Bütün sıralı olarak verildiği ve herhangi bir sonuçun üretilmediği bir operasyondur.

Foreach stream operasyonu parametre olarak Consumer veya Bi Consumer almaktadır. Kaynak içerisindeki elemanların tek tek Consumer’e verilir. Foreach operasyonu Terminate operasyon olduğundan Stream döngüsünü sonlandırır.

Foreach yapısı kullanılarak Stream içerisindeki her eleman ekrana yazdırılarbilir veya üzerlerinde işlem yapılabilir. En çok kullanıldığı yerlerden birisi genellikle ekrana yazdırma olmaktadır.

**Min and max**

Terminate operasyonlardan olan min ve max, kaynak içerisindeki en büyük ve en küçük elemanları döndürmektedir. Burada dikkat edilmesi gerekenlerden birisi Primitive Streamler içerisinde parametre almamaktadır çünkü iki farklı integer değerinden hangisinin büyük olduğuna karar verilebilir.

Nesnelerimizi sakladığımız Stream içerisinde ise Comparator interface’ini parametre olarak almaktadır. Yani min ve max kullanılırken nesnelerin nasıl kıyaslanacağına dair bir bilgi vermemiz gerekmektedir.

Min ve Max Operasyonu direk değişken vermek yerine Optional nesnesi döndürmektedir. Yani Stream kaynağımızda hiç bir verinin bulunmaması durumunda null değer döndürmemesi amaçlı olarak bu yapı kullanılmıştır.

**AnyMatch**

AnyMatch Stream API Operasyonu kaynak içerisinde bulunan ve kriterlere uyan herhangi bir elemanın olup olmadığın kontrol eder. Parametre olarak Predicate alır ve buna göre kaynak üzerinde koşulu sağlayan herhangi bir elemanın bulunması üzerine true sonuç döndürmektedir.

AnyMatch kullanarak kaynak içerisinde varolmasını istemediğimiz bir verinin olup olmadığını kontrol edeibliriz.

**AllMatch**

AllMatch Stream API Operasyonu stream kaynağı içerisinde bulunan her bir verinin belirtilen koşulu sağlayıp sağlamadığını kontrol etmektedir. Örnek olarak bir veri listemiz içerisinde bütün elemanların 0dan büyük olup olmadığını kontrol edebiliriz.

AllMatch parametre olarak Predicate alır ve bu koşulu bütün elemanlar üzerinde dener, herhangi bir elemanın koşula uymaması durumunda false döner, bütün elemanlar koşulu sağlıyor ise true sonuç vermektedir.

Stream API noneMatch Operasyonu kaynak içerisinde bulunan bütün elemanların verilen koşula uymaması gerekmektedir, eğer herhangi bir eleman bu koşula uyuyor ise false sonucu döner.

**NoneMatch**

NoneMatch operasyonu kaynak içerisinde ki hiçbir elemanın bu koşula uymaması anlamını taşımaktadır. Allmatch ile farkı ise bu noktadadır, allmatch bütün elemanların verilen koşula uyması gerekmekteydi.

NoneMatch Stream Operasyonu aldığı Predicate Functional Interface’i ile kaynaktaki bütün elemanları kontrol eder eğer herhangi bir eleman koşula uyuyor ise false sonucu verir, eğer hiçbir eleman sonuca uymuyor ise true sonuç vermektedir.

**Reduce**

Java 8 Stream reduce operasyonu kaynaktaki her elemanın birleştirilerek tekil bir eleman oluşturulması işlemini görmektedir, yani eğer bir String listesini kaynak olarak kullanıyorsak bu Stringlerin birleştirilip bir adet String değişkeni üretmemiz reduce operasyonu olarak adlandırılır.

Reduce yapabilmek için iki farklı parametreyi belirtmek zorundayız. İki parametreden ilki olan ilk değer parametresi birleştirme işlemi yaparken başlangıç değerini belirtmektedir. Bu başlangıç değeri kaynaktaki ilk eleman ile birleştirilir.

İkinci bir diğer parametremiz ise BinaryOperator tipinde olmalıdır, burada yapılan işlemi belirtirirz. Kaynak içinde yer alan verilerin nasıl birleştiriliceğine dair bir fonksiyon yazarız.

Reduce Stringleri birleştirmek için veya kaynak içerisindeki elemanların toplanması için kullanılabilmektedir.

**Primitive Tipler**

IntStream intStream = IntStream.range(1,3);

IntStream rangeClosed = IntStream.rangeClosed(1,3);

IntStream iterate = IntStream.iterate(1, i -> i+2).limit(4);

IntStream i3 = Arrays.stream(new int[] {1,2,3});

IntStream i4 = new Random().ints().limit(4);

**Range**

Belirtilen sayılar arası değer üretir. Açık aralıktır. Son değeri dahil etmez.

**RangeClosed**

Belirtilen sayılar arası değer üretir. Kapalı aralıktır. Son değeri dahildir.

**Average**

Elemanların ortalamasını verir. OptionalDouble dönmektedir.

**summaryStatistics**

Stream’ın istatistik bilgilerini tutar.

**WORKING WITH ADVANCED SYSTEM PIPELINE CONCEPT**

AdvancedStream sınıfındaki örnekte bu konu açıklanmıştır.

Streamler lazy yapıda olduğu için, terminal operatör çağırıldığı zaman işleme alınacağından listeye sonradan eklenen değer, stream’a de eklenecektir.

**Collect**

**Şimdiye kadar birçok şekilde stream oluşturma örneği içerisinde değişken türlerinin Stream kaynağı olarak nasıl kullanılacağını gördük. Tam tersi bir şekilde, Stream kaynağını farklı değişken türlerine çevirebiliriz.**

**Bu işlem için java.util.stream paketinde yer alan Collectors sınıfı kullanılmaktadır**.

**Collectors sınıfı içerisine çeşitli önceden tanımlanmış işlemler yer almaktadır.** Aşağıda bu işlemlerin bir kısmı yer almaktadır.

* [averaging](https://docs.oracle.com/javase/8/docs/api/java/util/stream/Collectors.html#averagingDouble-java.util.function.ToDoubleFunction-)
* counting
* [grouping](https://docs.oracle.com/javase/8/docs/api/java/util/stream/Collectors.html#groupingBy-java.util.function.Function-)
* [joinin](https://docs.oracle.com/javase/8/docs/api/java/util/stream/Collectors.html#joining--)g
* mapping
* maxBy, minBy
* [partitioning](https://docs.oracle.com/javase/8/docs/api/java/util/stream/Collectors.html#partitioningBy-java.util.function.Predicate-)
* reducing
* summarizing
* summing
* toCollection
* toCouncurrentMap
* toList
* toMap
* toSet

**Joining**

Stream’de belirtilen karakter veya Stringleri birleştirip geriye String olarak dönüş sağlar.

**groupingBy**

Belirtilen koşula göre , gruplama yapıp default olarak List tipinde dönüş sağlar. Geri dönüş tipini farklı seçenekte belirtebiliriz.

**partitioningBy**

Koşula göre şart sağlanıyorsa sağlananları true, sağlanmayanları false listesine atar.

**toList**

Stream’ı listeye çevirir.

**toSet**

Stream’ı listeye çevirir.

**toMap**

Stream’ı map’e çevirir.

**toCollection**

Herhangi bir class vermek istediğimiz zaman kullanılır.

**Sum**

Elemanların toplamını alır.

**Average**

Elemanların ortalamasını verir

**Optional**

Optional sınıfları bir veri tipinin null olup olmasını kontrol edebilmek amaçlı oluşturulmuştur. Optional nesnesi üzerinden tanımlandığı veri tipindeki değişkenin var olup olmadığını kontrol edebiliriz.

Optional Metotları:

**get**()-> Optional içerisindeki değişkene erişmek için kullanılır.

**ifPresent**() -> Data varsa consumer olarak verilen işlemi yaptırır.

**isPresent**() -> Optional nesnesinin boş olup olmadığını kontrol eder. Veri varsa true, yoksa false döner.

**orElse**() -> Stream’de istenilen koşul sağlanmadığında yapması gereken işlemi belirtiriz.

**orElseThrow**-> Stream’de istenilen koşul sağlanmadığında tanımladığımız exception’ı fırlatır.