Link: **https://www.youtube.com/playlist?list=PLeZr8VTNC1oagAZy1XxF987E-F26xuCgP**

Lambda-01 09.11.2020

"Üşenme, Erteleme, Vaz geçme!"

Java 8 ile birlikte gelen Lambda ifadeler anonim fonksiyonlardır. Yani herhangi bir ismi olmayan methodlardır. Bunun yanı sıra parametre alabilir, method gövdesine sahip olabilir ve bir değer döndürebilir.

Temel olarak methodun iş yapan mantık kısmına odaklanmaya ve kod kalabalığından kurtulmaya yarar diyebiliriz.

1) Lambda "Functional Programming" dir. "Functional Programming" de "Nasıl yaparım?" değil "Ne yaparım?" düşünülür.
2) "Structured Programming" de "Ne yaparım?" dan çok "Nasıl yaparım?" düşünülür.
3) "Functional Programming" hız, kod kısalığı, kod okunabilirliği ve hatasız kod yazma (güvenilirlik) açılarından çok faydalıdır.
4) Lambda sadece **Collection**'larda (List, Queue ve Set) ve **Array**'lerde kullanılabilir.

! Lambda'yi aşağıdaki gibi farklı sahalarda kullanabiliriz;
   1. İnteger List,
   2. String List,
   3. Object List (Course Object),
   4. File (text),
   5. IntStream (Structured Programming'de for-Loop kullanımı gibi).

- Functional => fonksiyon demek. Programlama dillerinde fonksiyon programın iş yapan kısmına denir. Java fonksiyon yerine method kullanır. "Functional Programming" method kullanma programlama dili demektir.
- Lambda 'da sürekli method kullanılır. Örneğin; For loop yazılmaz, for loop içeren method kullanılır. Elemanları sıralamayacağız, sıralayan methodu kullanacağız. En büyük elemanı bulmak için kod yazmayacağız, ilgili methodu kullanacağız.
- Lambda 'da kod yazmazsınız, method seçip kullanırsınız. Kod yazmak hoş karşılanmaz.
- Structured => yapı demek.

stream() => Bu koleksiyonun (List elemanlarını) kaynağı olarak sıralı bir Akış döndürür. Yukarıdan aşağı dizildiğinde yatay dizilmeye göre daha fazla method kullanılır. forEach() hem yatay hem de dikeyde çalışır ama bazı methodlar ya sadece yatay yada sadece dikeyler için çalışıyor. List'in kendisine ait bazı methodlar sadece yatayda çalışır.

forEach() => "İngilizce, her birisi için" anlamındadır. Bu akışın her bir öğesi için bir eylem gerçekleştirir. Dikey datalar bitene kadar çalışır.

t-> => Lambda Operatörü. Burada genellikle **t** harfi sonrasında **u** ve **v** harfi kullanılır.

- **t->** System.out.print(t + " ") => Burası **Lambda Expression**'dir tavsiye edilmez, Method Reference tavsiye edilir. (System.out**::**print gibi)
- **Method Reference** bir methoda yönlendirmek demektir. Class ismi **::** Method ismi şeklinde yazılır.

filter() => Bu akışın verilen yüklemle eşleşen öğelerinden oluşan bir akış döndürür. Filtreleme işlemi gerçekleştirir. Bir liste veya dizi içerisinde bazı şartlara göre elemanları ayırt etmek isteyebiliriz, mesela bir sipariş listesinde fiyatı en yüksek olan siparişi getir veya kullanıcı listesinde yaşı belirli bir yaşın üzerinde olanları getir diyebiliriz. Kısaca bir listeyi; bir veya birden çok parametreye göre kısıtlayabiliriz.

map() => Verilen işlevi bu akışın öğelerine uygulamanın sonuçlarından oluşan bir akış döndürür. Bu bir ara işlemdir. Stream içerisinde erişilen her bir nesneye özgü işlemler yapmamızı sağlar. Lambda'da update etmek değişiklik yapmak için map() methodu kullanılır. İçine **Lambda Expression** yazılır.

Math => Math Class'i, temel üstel, logaritma, karekök ve trigonometrik fonksiyonlar gibi temel sayısal işlemleri gerçekleştirmek için yöntemler içerir.

sqrt => Çift değerin doğru şekilde yuvarlanmış pozitif **karekökünü** verir.

reduce() => Azaltmak demek. Reduce işlemi genelde kümülatif operasyonlarda sıkça kullanılır. Bir veri setinde sırayla işlem yapmak istiyorsanız ve bir önceki yaptığınız işlemi de dahil etmek istiyorsanız reduce metodunu kullanabilirsiniz. İlişkili bir biriktirme işlevi kullanarak bu akışın öğelerinde bir azalma gerçekleştirir ve varsa indirgenmiş değeri açıklayan bir İsteğe Bağlı döndürür. Maksimum-minimum bulma, toplama-çarpma işleminde kullanılır.

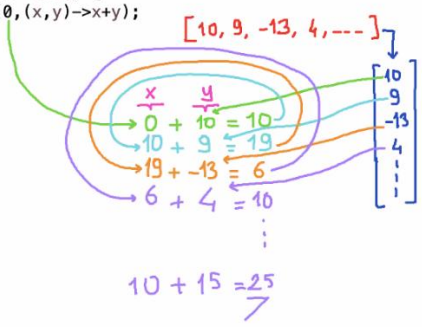
max => İki int değerinden büyük olanı döndürür. Yani, sonuç, Integer.MAX_VALUE değerine daha yakın olan bağımsız değişkendir. Bağımsız değişkenler aynı değere sahipse, sonuç aynı değerdir.

Optional<Integer> => Bir Class'tır. Boş olmayan bir değer içerebilen veya içermeyen bir kap Objecttir. Bir değer varsa, isPresent () true, get () ise değeri döndürecektir. Ek olarak, bu sınıf, bir int'i bir String'e ve bir String'i bir int'e dönüştürmek için çeşitli yöntemler ve ayrıca bir int ile ilgilenirken yararlı olan diğer sabitler ve yöntemler sağlar. Exception gibi çalışır.

Lambda-02 10.11.2020

List 'teki tüm elemanların toplamını alırken; **reduce(0, (x,y) -> x+y);** yada **reduce(0, Integer::sum);** kullanılabilir.

```
//List'teki tum elemanlarin toplamini console'a yazdiriniz
public static void toplamEl(List<Integer> list) {
    int toplam  = list.stream().reduce(0,(x,y)->x+y);
    System.out.println(toplam);
}
```



sorted() => Doğal düzene göre sıralanmış, bu akışın öğelerinden oluşan bir akış döndürür. Sorted() methodu tekrarlı kullanılırsa en son kullanılan aktif oluyor.

Comparator => bir Class'tır. Compar karşılaştırmak demektir. Bazı nesneler koleksiyonuna toplam sıralama uygulayan bir karşılaştırma işlevi. Karşılaştırıcılar, sıralama düzeni üzerinde hassas kontrol

sağlamak için bir sıralama yöntemine (Collections.sort veya Arrays.sort gibi) aktarılabilir. Karşılaştırıcılar, belirli veri yapılarının (sıralı kümeler veya sıralı haritalar gibi) sırasını kontrol etmek veya doğal sıralaması olmayan nesnelerin koleksiyonları için bir sıralama sağlamak için de kullanılabilir.

reverseOrder() => Comparator Class'ının bir methodudur. Doğal sıralamanın tersini (büyükten küçüğe) uygulayan bir Comparator (karşılaştırıcı) döndürür.

comparing() => karşılaştırma demektir.

reversed() => Bu karşılaştırıcının (comparator) ters sıralanmasını uygulayan bir karşılaştırıcı (comparator) döndürür.

Lambda-03 11.11.2020

distinct() => Bu method tekrarlı elemanları sadece bir kere yazdırır. Bu akışın farklı öğelerinden (Object.equals (Object) 'e göre) oluşan bir akış döndürür. Sıralı akışlar için, farklı öğelerin seçimi sabittir (yinelenen öğeler için, karşılaşma sırasında ilk görünen öğe korunur.) Sırasız akışlar için, herhangi bir kararlılık garantisi verilmez. Stream return eder.

Match operasyonu bir akışın belirli kriterleri sağlayıp sağlamadığını ölçmek için kullanılır. Map den farkı her iterasyonu tek tek değerlendirip sonucu yansıtmaz bunun yerine tüm koleksiyonu değerlendirerek sonucu yansıtmasıdır. Match operasyonunun 3 çeşit kullanımı bulunmaktadır;

allMatch() => "**Hepsi** uyar, vereceğim kurala" demektir. Bu akışın **tüm öğelerinin** sağlanan yüklemle eşleşip eşleşmediğini döndürür. Sonucu belirlemek için gerekli değilse tüm öğelerdeki yüklemi değerlendirmeyebilir. Akış boşsa, **true** döndürülür ve dayanak değerlendirilmez.

noneMatch() => **Hiç biri** uymaz. Bu akışın **hiçbir** öğesinin sağlanan yüklemle eşleşip eşleşmediğini döndürür. Sonucu belirlemek için gerekli değilse tüm öğelerdeki yüklemi değerlendiremeyebilir. Akış boşsa, **true** döndürülür ve dayanak değerlendirilmez.

anyMatch() => Bu akışın **herhangi bir** öğesinin sağlanan öngörü ile eşleşip eşleşmediğini döndürür. Sonucu belirlemek için gerekli değilse tüm öğeler üzerindeki yüklemi değerlendirmeyebilir. Akış boşsa, **false** döndürülür ve yüklem değerlendirilmez.

findFirst() => İlk elemanı verir. Bu akışın **ilk öğesini** açıklayan bir İsteğe Bağlı veya akış boşsa boş bir İsteğe Bağlı döndürür. Akışın karşılaşma sırası yoksa herhangi bir öğe iade edilebilir. Optional return eder.

limit(1) => Sınırlandırma demek. Bu akışın öğelerinden oluşan, uzunluğu maxSize'dan uzun olmayacak şekilde kesilmiş bir akış döndürür. **Stream return eder**.

- Stream'ler ekrana direk yazdırılamaz. Stream'i **toArray()** ile Array'e çevirirz. Array'i de **Arrays.toString()** 'in içine alıp yazdırabiliriz. Ör; System.out.println(Arrays.toString(***.toArray())); veya System.out.println(Arrays.asList(***.toArray())); kullanılabilir.

skip(1) => atlama demek. Akışın ilk n elemanını attıktan sonra bu akışın kalan elemanlarından oluşan bir akış döndürür. Bu akış n'den daha az öğe içeriyorsa, boş bir akış döndürülür. Bu, durum bilgisi olan bir ara işlemdir.

skip(list.size()-1) => List'in uzunluğunun 1 eksiğini yazarsak son elemanı yazdırırız.

collect() => collect topla demektir. Elemanları List'in içine toplamak için kullanılır. Sunucunuzu List içinde görmek için kullanırsınız. Bir Toplayıcı kullanarak bu akışın öğeleri üzerinde mutable reduction işlemi gerçekleştirir. Bir Toplayıcı, toplamak için argümanlar olarak kullanılan işlevleri (Supplier, BiConsumer, BiConsumer) kapsüller ve çok seviyeli gruplama veya bölümleme gibi toplama stratejilerinin yeniden kullanımına ve toplama işlemlerinin kompozisyonuna izin verir.

Akış paralelse ve Toplayıcı eşzamanlıysa ve akış sıralı değilse veya toplayıcı sıralanmamışsa, eşzamanlı bir azaltma gerçekleştirilecektir (eşzamanlı azaltma ile ilgili ayrıntılar için Toplayıcıya bakınız.)

Bu bir terminal işlemidir. Paralel olarak yürütüldüğünde, değişken veri yapılarının izolasyonunu sürdürmek için çoklu ara sonuçlar somutlaştırılabilir, doldurulabilir ve birleştirilebilir. Bu nedenle, iş parçacığı açısından güvenli olmayan veri yapılarıyla (ArrayList gibi) paralel olarak yürütüldüğünde bile, paralel bir azaltma için ek eşitleme gerekmez.

Collectors.toList() => Öğeleri koleksiyonlarda toplamak, öğeleri çeşitli kriterlere göre özetlemek gibi çeşitli yararlı azaltma işlemlerini uygulayan Collector Uygulamaları.

toList() => Giriş öğelerini yeni bir Listede toplayan bir Toplayıcı döndürür. Döndürülen Listenin türü, değiştirilebilirliği, serileştirilebilirliği veya iş parçacığı güvenliği konusunda hiçbir garanti yoktur; döndürülen Liste üzerinde daha fazla denetim gerekiyorsa, toCollection (Tedarikçi) kullanın.

Lambda-04 12.11.2020

mapToInt() => map() methodu **integer** kullanmadığı için toplama yapamayız. Verilen işlevi bu akışın öğelerine uygulamanın sonuçlarından oluşan bir IntStream döndürür. Bu bir ara işlemdir.

sum() => Bu akıştaki öğelerin toplamını döndürür.

mapToDouble() => **Ondalık sayılarda** kullanılır. Verilen işlevi bu akışın öğelerine uygulamanın sonuçlarından oluşan bir DoubleStream döndürür.

OptionalDouble => Çift değer içerebilen veya içermeyen bir konteyner nesnesi.

average() => Bu akışın elemanlarının **aritmetik ortalamasını** açıklayan bir İsteğe Bağlı Çift döndürür veya bu akış boşsa isteğe bağlı boş bir değer döndürür. Döndürülen ortalama, değerlerin kaydedildiği sıraya bağlı olarak değişebilir. Bu yöntem, ortalamayı hesaplamak için kullanılan sayısal toplamdaki hata sınırını azaltmak için telafi edilmiş toplama veya başka bir teknik kullanılarak uygulanabilir. Mutlak büyüklük arttıkça sıralanan öğeler daha doğru sonuçlar verme eğilimindedir. OptionalDouble return eder.

max() => Bu akışın **maksimum** elemanını açıklayan bir OptionalDouble döndürür. Sayısal karşılaştırma işleçlerinden farklı olarak, bu yöntem negatif sıfırın kesin olarak pozitif sıfırdan daha küçük olduğunu düşünür.

min() => Bu akışın **minimum** öğesini açıklayan bir OptionalDouble döndürür.

mapToLong() => Astronomi gibi **çok uzun sayıları**n olduğu yerlerde **kullanılır**.

count() => Bu akıştaki **öğelerin sayısını** döndürür. Return type'i **long döndürür**.

IntStream => Lambda Collection'larla calisir. Collection kullanmadığımız zaman IntStream'e **range()** methodu ile sınırlar verildiğinde sınırları bir List gibi kullanır. Bir aralık ister, o aralığı da **range()** methodu verir. Sıralı ve paralel birleştirme işlemlerini destekleyen primitive, int değerli öğeler dizisi.

Bu, Stream'in sezgisel primitive uzmanlaşmasıdır. Akışların, akış işlemlerinin, akış işlem hatlarının ve paralelliğin ek özellikleri için Stream için sınıf belgelerine ve java.util.stream paket belgelerine bakın.

range(int **startInclusive**, int **endExclusive**) => Başlangıç parametresi dahil, bitiş parametresi hariçtir. "startInclusive" (dahil) ile "endExclusive" (hariç) arasında sıralı bir IntStream'i 1'lik artımlı bir adımla döndürür. Eşdeğer bir artan değerler dizisi, bir for döngüsü gibi kullanılarak sırayla üretilebilir. range(1, x+1) şeklinde yazılır.

rangeClosed(int **startInclusive,** int **endInclusive)** => "StartInclusive (inclusive)" ile "endInclusive (inclusive)" arasında artımlı 1 adımı ile sıralı bir IntStream döndürür. rangeClosed(1, x) şeklinde yazılır.

iterate(seed, f) => Bir f fonksiyonunun bir ilk öğe (seed) yinelemeli uygulamasıyla üretilen sonsuz sıralı IntStream döndürür. limit() kullanmak gerekebilir.

- Hücre sayısı, atom sayısı vb. gibi çok büyük sayılarla çalışabilmek için;

mapToObj() => Kullanılacak sayılar integer ve long u aşacaksa mapToObj() methodu kullanılır. Verilen sayıları Object'e dönüştürür. Verilen işlevin bu akışın öğelerine uygulanmasının sonuçlarından oluşan Object değerli bir Akış döndürür.

BigInteger => Büyük sayıların depolandığı bir Class'tır. BigInteger, Java'nın tüm primitive integer operatörlerine ayrıca modüler aritmetik, GCD hesaplaması, asallık testi, ana üretim, bit işleme ve birkaç başka çeşitli işlem için işlemler sağlar.

valueOf => Değeri belirtilen uzunluktaki değere eşit olan bir BigInteger döndürür. Bu "static factory method ", bir (uzun) kurucuya tercih edilir, çünkü sık kullanılan BigInteger'ların yeniden kullanımına izin verir.

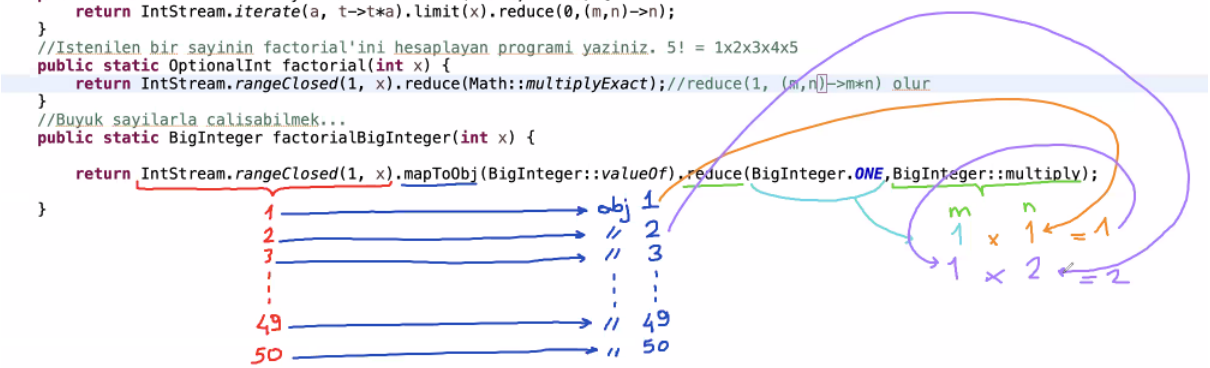
ONE => BigInteger sabiti olan. Büyük sayılarla çalışırken 1 gibi integer sayı kullanmaya Java itiraz eder. Bunun yerine kullanılır. Final variable olduğu için büyük harflerle yazılır.

multiply => Bir uygulama **val == this** olduğunda daha iyi algoritmik performans sunabilir.

```
//Istenilen bir sayinin istenilen kuvvetini ekrana yazdiran programi yaziniz
public static int istenenSayininIstenenKuvvet(int a, int x) {
    return IntStream.iterate(a, t->t*a).limit(x).reduce(0,(m,n)->n);
}
//Istenilen bir sayinin factorial'ini hesaplayan programi yaziniz. 5! = 1x2x3x4x5
public static OptionalInt factorial(int x) {
    return IntStream.rangeClosed(1, x).reduce(Math::multiplyExact);//reduce(1, (m,n)->m*n) olur
}
//Buyuk sayilarla calisabilmek...
public static BigInteger factorialBigInteger(int x) {

    return IntStream.rangeClosed(1, x).mapToObj(BigInteger::valueOf).reduce(BigInteger.ONE,BigInteger::multiply);

}
```



- Java'da File oluşturmak için package üstüne sağ tıklanır, New içinde "Untitled Text File" seçilir. (Eğer "Untitled Text File" görülmezse en altta Other tıklanır, General içindeki "Untitled Text File" seçilir.) İçine istenilen text (metin) yazılır. File tıklanır ve Save As ya da Save All seçilir. Gelen pencerede "File name"e istenilen dosya ismi (FileForLambda) yazılır, File'ı kaydetmek istediğimiz location belirlenir ve OK tıklanır.

Files => Dosya okumak için kullanılan bir Class'tır. Bu sınıf, yalnızca dosyalar, dizinler veya diğer dosya türleri üzerinde çalışan statik yöntemlerden oluşur. Çoğu durumda, burada tanımlanan yöntemler, dosya işlemlerini gerçekleştirmek için ilişkili dosya sistemi sağlayıcısına yetki verir.

lines(path) => Bir dosyadaki (Files) tüm satırları (lines) Akış olarak okuyun. Dosyadaki baytlar, UTF-8 karakter kümesi kullanılarak karakterlere dönüştürülür. Bu yöntem, onu çağırmak, ifadeyi değerlendirmeye eşdeğermiş gibi çalışır: Stream return eder. Path yerine => lines(Paths.get("src/lambda/FileForLambda")) yazılır. Burada Javaya "throws IOException" atmak gerekir. Ekrana yazdırmak için "forEach(System.out::println);" kullanılır.

"FileForLambda" dosyasını ekrana büyük harflerle yazdırmak için "map(String::toUpperCase)" methodu kullanılır.

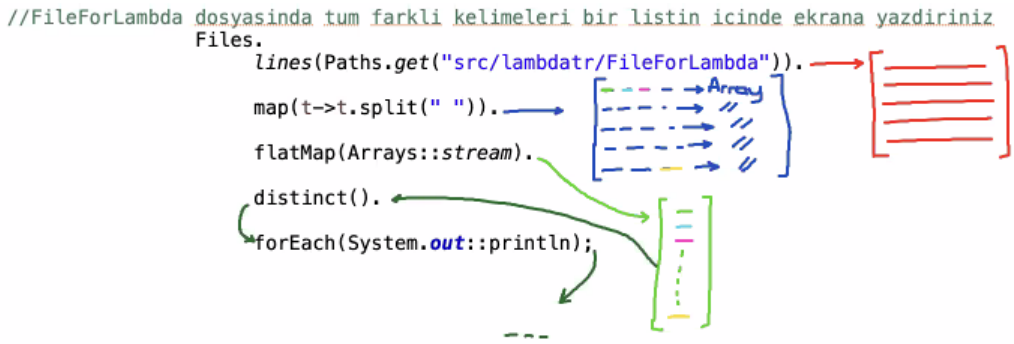
File içinde yapılan her değişiklik Save edilmeli.

Lambda-05 13.11.2020

split() => Bu String öğesini verilen normal ifadenin eşleşmelerine **böler**. Bu yöntem, iki bağımsız değişkenli bölünmüş yöntemi verilen ifade ve sıfır sınır değeriyle çağırır gibi çalışır. Sondaki boş dizeler bu nedenle sonuç dizisine dahil edilmez.

flatMap() => Herhangi bir Collection'daki (Array gibi) elemanları birer birer Stream'e yerleştirir. flatMap()'i kullanabilmek için elimizde bir Collection olmak zorunda.

Bu stream'in her bir öğesinin, sağlanan eşleme işlevinin her öğeye uygulanmasıyla üretilen eşlenmiş bir akışın içeriğiyle değiştirilmesinin sonuçlarından oluşan bir akış döndürür. Eşlenen her akış, içeriği bu akışa yerleştirildikten sonra kapatılır. (Eşlenmiş bir akış **null** ise bunun yerine boş bir akış kullanılır.) flatMap() işlemi, akışın öğelerine bire çok dönüşüm uygulama ve ardından ortaya çıkan öğeleri yeni bir akışa düzleştirme etkisine sahiptir. flatMap(Arrays::stream) şeklinde kullanılır.



Yukarıdaki örnekte;

Lines() => satirlardan (line) oluşan bir Stream oluşturur,

map()'in içindeki split() => Stream'i boşluklardan ayırarak Array'e çevirir,

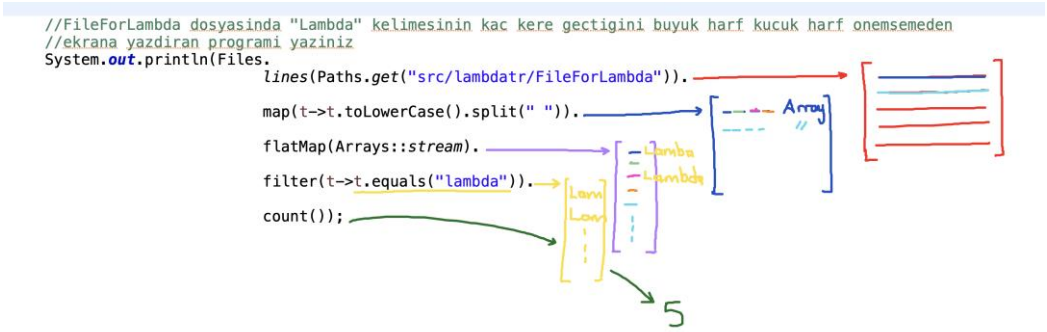
flatMap() => bir Collection'daki (Array gibi) elemanları birer birer Stream'e yerleştirir,

distinct() => tekrarlı kelimeleri bire indirir,

forEach() => Console'da ekran çıktısı (output) alınır.

Eğer **List**'in içine yazdırmak istersek forEach() yerine collect(Collectors.toList()) methodu kullanırız.

Başka bir örnek için aşağıdaki resim incelenebilir;



Eclipse'de yazılan kodları IntelliJ'e aktarmak için; **https://youtu.be/Pb5zGZuf45A** videosuna bakılabilir.