What are streams?

It is introduced in Java 8. It is used to apply aggregate operation on the data returned from collection classes and helps to achieve code optimization i.e., reduce the line of code into a single line.

Lambda Expression:

It introduces new arrow operator -> and splits the expression into 2 parts where the left side of it hold zero or one or more than one parameter and right side will be to apply the action that the user wanted to perform.

**How to achieve in 3 steps:**

1. Convert the list / set into streams by saying respective object name.stream()
2. Apply an intermediate operator like filter() to transform into another stream and so on.
3. Apply terminal operator like count() on the final stream to get the result.

**Example to understand the implementation of stream:**

**Using Filter methods**

The user needs to identify the number of object in the list of string that starts with “A”.

ArrayList<String> list = new ArrayList<>():

List.add(“Amala”);

List.add(“Bujacko”);

List.add(“Sharanya”);

List.add(“Anoka”);

//below line of codes is required to achieve the operation.

**Int count=0;**

**For(int i=0;i<list.size();i++){**

**If(list.get(i).startsWith(“A”)){**

**Count++:**

**}**

**Sysout(“The count of strings that starts with A is : “+count);**

**//but with streams**

Long c = list.stream().filter(s->s.(startsWith(“A”)).count();

**Sysout(“The count of strings that starts with A is : “+c);**

**Note:**

1. **count is the terminal operator and return type is long. It will get executed only if the filter methods returns true, else it won’t.**
2. **Without count method or any other terminal operator, any lambda expression is useless.**
3. **Filter is the intermediate operator.**

**Stream method will work only on stream**

So instead of creating a list / set and converting them into stream, a short cut would be to create them directly as light weight streams.

Stream.of(“Name1”,”Name2”,”Name3”).filter(s->s.startsWith(“A”)).count();

**Note**: in both the methods, if the right part of the arrow has more than one line then it can be written inside a flower bracket as like method.

Long d = Stream.of(“Name1”,”Name2”,”Name3”).filter(s->

{

s.startsWith(“A”));

return false; //purposely making the stream to return false to see count() won’t work

}).count();

Sysout(d); //o/p is zero

**How to print the objects in the list:**

list.stream().filter(s->s.length()>4).forEach(s->System.out.println(s));

filter(s->s.length()>4) 🡪 filtering the object from stream where the length is greater than 4 and store into a variable s

forEach(s->System.out.println(s)) – on the new stream iterating and printing all objects.

**To limit the output to one instead of multiple values:**

list.stream().filter(s->s.length()>4).limit(1).forEach(s->System.out.println(s));

**using Maps:**

When you want to perform some modifications on the filtered stream, map can be useful. Let us say after you filter the object that starts with A and you want to print them in upper case, then map will be useful.

list.stream().filter(s->s.endsWith(“a”)).map(s->s.toUpperCase()).forEach(s->System.out.println(s));

Note: If you want to print all objects in list, then don’t use filter on streams instead directly call map and for each.

**How to print the output in sorted form:**

list.stream().filter(s->s.startsWith(“A”)).sorted().map(s->s.toUpperCase()).forEach(s->system.out.println(s));

**How to merge 2 different list named list and list1:**

Stream<String> newlist = Stream.concat(list.stream(), list1.stream());

**How to print the new list sorted:**

newlist.sorted().forEach(s->sysout(s));

**Match method in stream:**

When you want to check whether the object contains a particular content. Let us say the user wanted to check whether it has “Sharanya” and if yes returns true.

Boolean flag = newList.stream().anyMatch(s->s.equalsIgnorecase(“Sharanya”));

Assert.assertTrue(flag);

**How to convert a stream into a list again:**

There are sometime where you want to modify the list and then perform the desired operation, then in that case we can make of collector interface which will convert the stream into list again.

ArrayList<String> al = Newlist.stream().filter(s->s.endsWith(“a”)).map(s->s.toUpperCase()).collect(Collectors.toList());

**How to print the unique values from a list:**

List<Integer> list = Arrays.asList(3,2,2,1,4,5,4,5,6,6);

list.stream().distinct().forEach(s->system.out.println(s));

**Terminal operators:**

Count() – returns the count of the object

limit() – returns the first element from the filtered list

sorted() – sort the output