**Java-8 – Various methods of Stream**

* **Processing by collect() method:**

This method collects the elements from the stream and add it to the specified Collection.

Example:

List<String> l = new ArrayList<>();

l.add(“Pavan”);

l.add(“RaviTeja”);

l.add(“Chiranjeevi”);

l.add(“Venkatesh”);

l.add(“Nagarjuna”);

System.out.println(l); [Pavan, RaviTeja, Chiranjeevi, Venkatesh, Nagarjuna]

List<String> l1 = l.stream().filter(s -> s.length >= 9).collect(Collectors.toList());

System.out.println(l1); [Chiranjeevi, Venkatesh, Nagarjuna]

// Create a new list with all the elements as upper case

List<String> l2 = l.stream().map(s -> s.toUpperCase()).collect(Collectors.toList());

System.out.println(l2);

// [PAVAN, RAVITEJA, CHIRANJEEVI, VENKATESH, NAGARJUNA]

* **Processing by count() method:**

This method returns the number of elements present in the Stream.

public long count()

Example:

List<String> l = new ArrayList<>();

l.add(“Pavan”);

l.add(“RaviTeja”);

l.add(“Chiranjeevi”);

l.add(“Venkatesh”);

l.add(“Nagarjuna”);

System.out.println(l); [Pavan, RaviTeja, Chiranjeevi, Venkatesh, Nagarjuna]

long count = l.stream().filter(s -> s.length() >= 9).count();

System.out.println(“The number of strings whose length >= 9:”+count);

* **Processing by sorted() method:**

We can use sorted() method to sort elements inside stream.

We can sort either based on default natural sorting order or based on our own customized sorting order specified by comparator object.

sorted() -> For default natural sorting order.

sorted(Comparator c) -> For customized Sorting order.

Example:

List<Integer> l = new ArrayList<>();

l.add(0);

l.add(10);

l.add(20);

l.add(5);

l.add(15);

l.add(25);

System.out.println(l); [0, 10, 20, 5, 15, 25]

// code for default natural sorting

List<Integer> l1 = l.stream().sorted().collect(Collectors.toList());

System.out.println(0, 5, 10, 15, 20, 25]

// code using comparator

List<Integer> l2 = l.stream().sorted((i1, i2) -> -i1.compareTo(i2)).collect(Collectors.toList());

System.out.println(l2); [25, 20, 15, 10, 5, 0]

* **Processing by min() & max() methods:**

min(Compartor c)

Returns minimum value according to specified comparator.

max(Compartor c)

Returns maximum value according to specified comparator.

Example:

List<Integer> l = new ArrayList<>();

l.add(0);

l.add(10);

l.add(20);

l.add(5);

l.add(15);

l.add(25);

System.out.println(l); [0, 10, 20, 5, 15, 25]

Integer min = l.stream().min((i1, i2) -> i1.compareTo(i2)).get();

System.out.println(“Minimum value:”+min);

Integer max = l.stream().max((i1, i2) -> i1.compareTo(i2)).get();

System.out.println(“Maximum value:”+max);

* **Processing by using forEach() method:**

This method won’t return anything.

This method can take Lambda expression as argument and apply that lambda expression for each element present in Stream.

Example:

List<String> l = new ArrayList<>();

l.add(“A”);

l.add(“BB”);

l.add(“CCC”);

// With lambda expression

l.stream().forEach(s -> System.out.println(s));

// With method reference

l.stream().forEach(System.out :: println);

* **Processing by toArray() method:**

We can use toArray() method to copy elements present in the Stream into specified array.

Example:

List<Integer> l = new ArrayList<>();

l.add(0);

l.add(10);

l.add(20);

l.add(5);

l.add(15);

l.add(25);

System.out.println(l); [0, 10, 20, 5, 15, 25]

Integer[] array = l.stream().toArray(Integer[] :: new);

for(Integer x: array){

System.out.println(x);

}

* **Streams for Arrays:**

**Stream.of() method**

We can also apply stream for group of values & for arrays.

For group of values:

Stream<Integer> s = Stream.of(9,99,999,9999,99999);

s.forEach(System.out :: println);

For Arrays:

Doulbe[] d = {10.0, 10.1, 10.2, 10.3, 10.4}

Stream<Integer> s1 = Stream.of(d);

s1.forEach(System.out :: println);

* **Summary of Streams:**

We have covered the following items in stream.

1. Stream s = c.Stream();
2. filter(Predicate<T> t);
3. map(Function<T, R> f);
   1. collect();
   2. count();
   3. sorted(Comparator c);
   4. min(Comparator c);
   5. max(Comparator c);
4. forEach();
5. Stream.of();

* **Note: This document contains contents of vidoes 82 to 86**