Q) What are streams?

- If we want to process bulk objects of collection then go for streams concept.
- Way to operate on collection in java 8 is Stream.
- Its a special iterator class that allows processing collections of objects in a functional manner.

- Eg : fetch all objects from collection of list whose value is greater than 15


```
6
7  public class StreamDemo {
8      public static void main(String[] args) {
9
10         List<Integer> arList =new ArrayList<Integer>();
11         arList.add(15);
12         arList.add(25);
13         arList.add(5);
14
15         List<Integer> newAl =new ArrayList<Integer>();
16
17         newAl = arList.stream().filter(x -> x>=15).collect(Collectors.toList());
18
19         newAl.stream().forEach(x -> System.out.println(x));
20
21         //without streams:
22         /*List<Integer> arListFromMEthod = findElements(arList);
23         for (Integer i : arListFromMEthod) {
24             System.out.println(i);
25         }
26     }
27
28     public static List<Integer> findElements(List<Integer> arList){
29         List<Integer> newAl =new ArrayList<Integer>();
```

Q) Difference between streams (java 1.8) and java.io.Stream?

- Why streams were introduced in java 8 if we already had java.io.stream?

- Java io streams is a sequence of characters or binary data which is used to be written to a file or to read data from a file.
- While streams java 1.8 is no where related to files, its related to collection object.
- Java io streams related to file whereas java 8 streams are related to collection object.
- Hence if we need to perform some operations on collection there we should go for streams.

Q) Difference between streams (java 1.8) and collection?

- To represent group of collection as single entity then we should use collection concept.
- If we want to perform operation on bulk objects in collection then we should go for Streams.

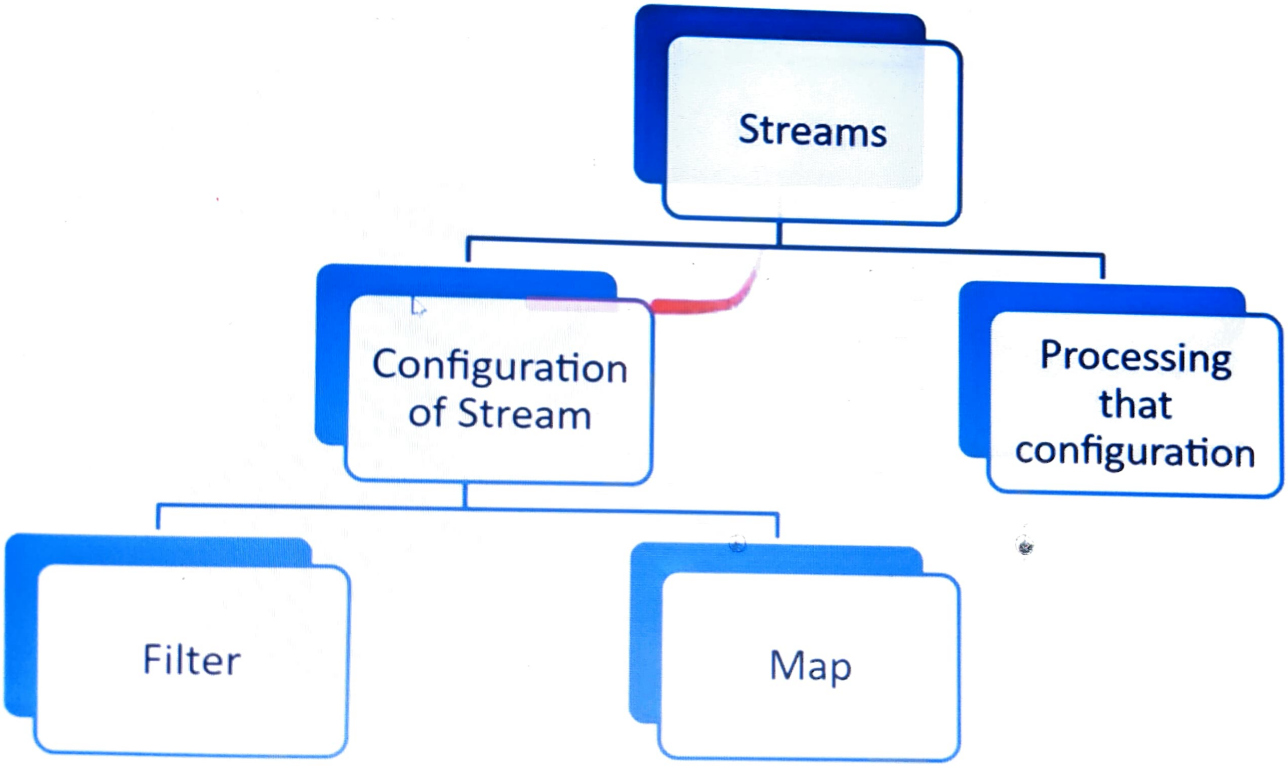
Q) Steps to create and process stream

- We can get stream object by :
  - Stream s = collectionObject.stream();
  - Once we get stream object we can process the object of collection.
  - Processing of stream consists of 2 steps/ stages
    - Configuration of stream
    - Processing that configuration

Configuration can be done by

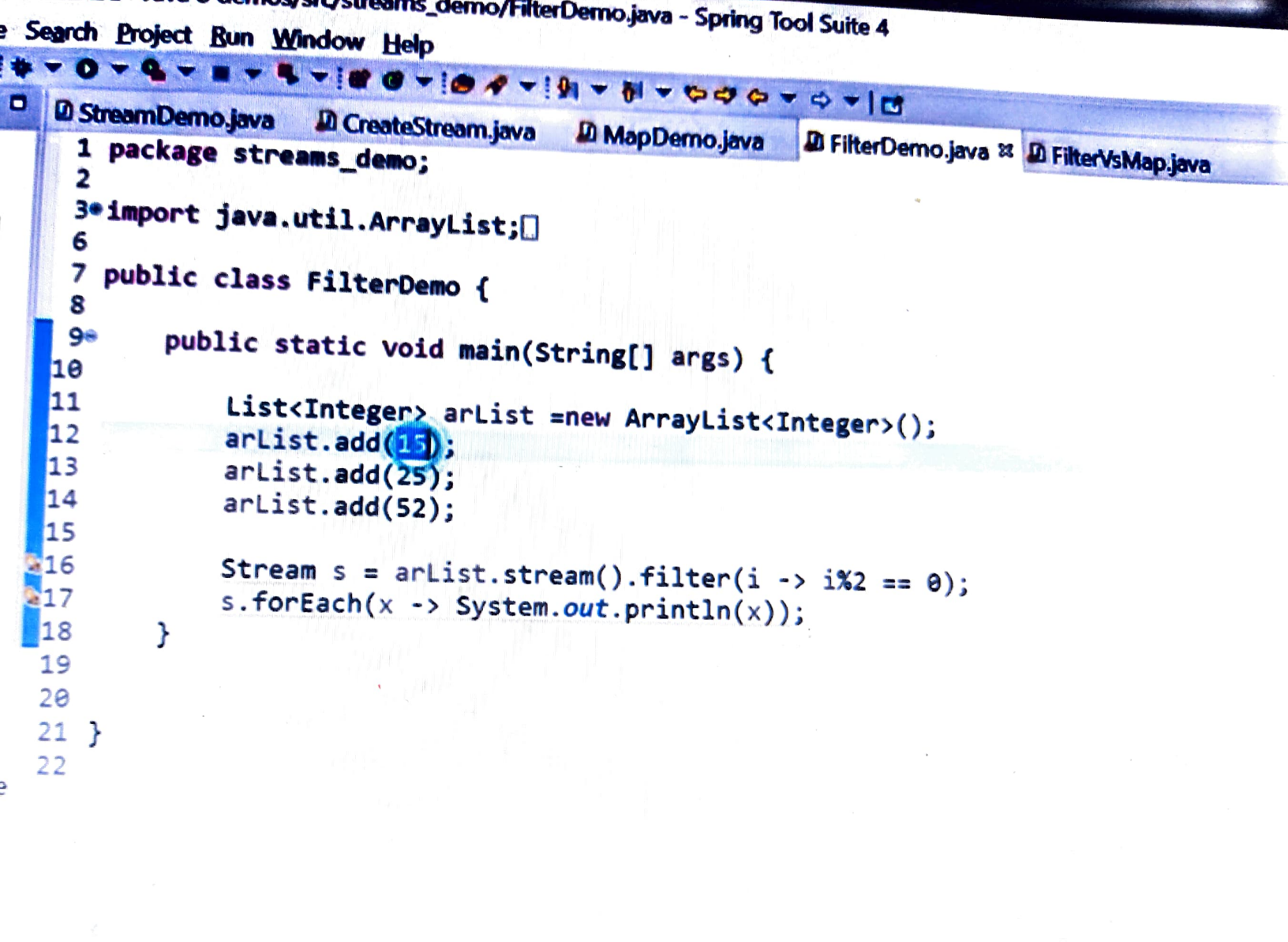
- Map
- Filter

Q) Steps to create and process stream



Q) How to Filter the stream objects

- Stream s = collectionObject.stream().filter(i → i % 2 ==0);
- See Demo


/src/streams_demo/FilterDemo.java - Spring Tool Suite 4
Search Project Run Window Help
StreamDemo.java
CreateStream.java
MapDemo.java
FilterDemo.java
FilterVsMap.java
1 package streams_demo;
2
3 import java.util.ArrayList;
6
7 public class FilterDemo {
8
9 public static void main(String[] args) {
10
11 List<Integer> arList =new ArrayList<Integer>();
12 arList.add(15);
13 arList.add(25);
14 arList.add(52);
15
16 Stream s = arList.stream().filter(i -> i%2 == 0);
17 s.forEach(x -> System.out.println(x));
18 }
19
20
21 }
22

Q) How to Map the stream objects

- What if we don't want to filter out.
- We rather want to create new object against each existing stream object based on some function.

- EG in given stream create new object by squaring its value
- Demo

```
2
3 import java.util.ArrayList;
6
7 public class MapDemo {
8
9     public static void main(String[] args) {
10
11         List<Integer> arList =new ArrayList<Integer>();
12         arList.add(15);
13         arList.add(25);
14         arList.add(52);
15
16         Stream s = arList.stream().map(i -> i*i);
17         s.forEach(x -> System.out.println(x));
18     }
19
20
21 }
```

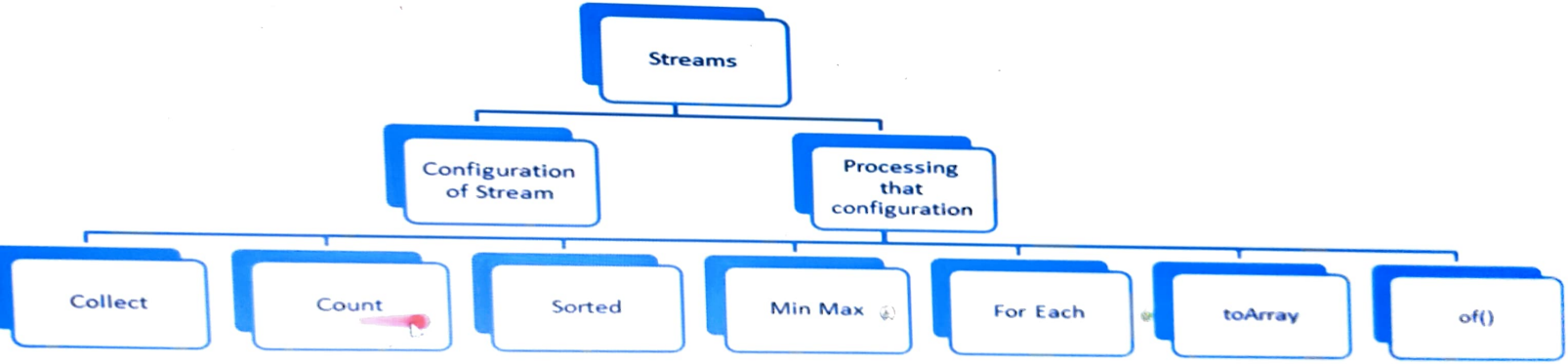
Q) Difference between Filter and Map

- If we want to fetch / filter objects from collection like eg : filter only even numbers from array list collection the use Filter for configuration of stream.

- If we want to perform some operation on each objects of the collection then create another mapped object with different value(after operation is performed) for each object of that collection, then use map.

- In filter, because of filtering, number of objects in filtered list is less than original list While in Map same number of objects are there in both new and original list created.

Q) Steps to create and process stream



Q) How to Process elements using collect()?

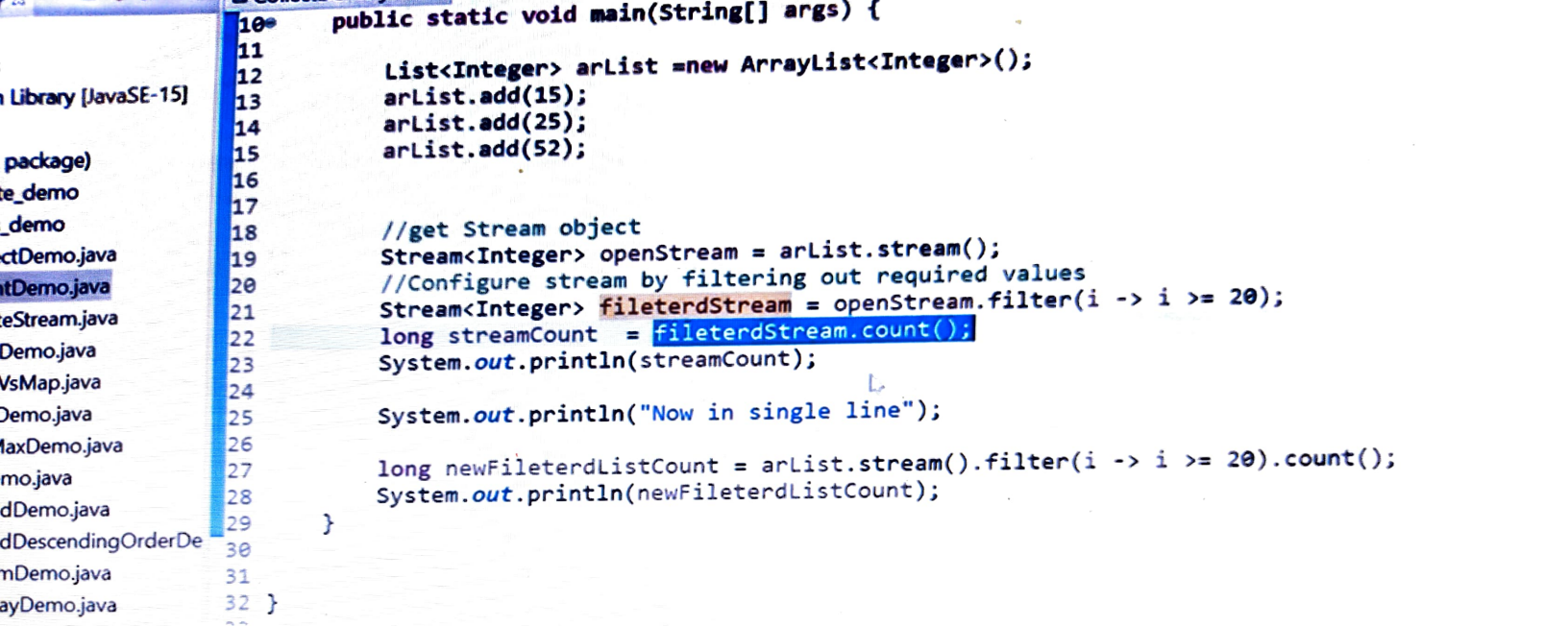
- If we want to collect elements of stream after filtering or mapping and add them to the required collection then use collect method.
- This will work for both filter and collect methods.
- Demo

CollectDemo.java

```
13          arList.add(15);
14          arList.add(25);
15          arList.add(52);
16
17
18          //get Stream object
19          Stream<Integer> openStream = arList.stream();
20          //Configure stream by filtering out required values
21          Stream<Integer> fileterdStream = openStream.filter(i -> i >= 20);
22          List<Integer> newFileterdListOrigin  = fileterdStream.collect(Collectors.toList());
23          newFileterdListOrigin.stream().forEach(x -> System.out.println(x));
24
25          System.out.println("Now in single line");
26
27                  // What we have done :
28                  // 1) Open stream with .stream()
29                  // 2) use lambda expression in filter to filter out required objects from the open stream.
30                  // 3) Collect all filtered elements and add them to a new list called newFileterdList
31
32
33          List<Integer> newFileterdList = arList.stream().filter(i -> i >= 20).collect(Collectors.toList());
34          newFileterdList.forEach(x -> System.out.println(x));
35      }
36
```

Q) How to Process elements using count()?

- If we want to count how many elements are there in collection that satisfy given condition then use collect method.
- Demo


Library [JavaSE-15]
package)
te_demo
_demo
ctDemo.java
tDemo.java
eStream.java
Demo.java
VsMap.java
Demo.java
MaxDemo.java
mo.java
dDemo.java
dDescendingOrderDe
mDemo.java
ayDemo.java
public static void main(String[] args) {
List<Integer> arList =new ArrayList<Integer>();
arList.add(15);
arList.add(25);
arList.add(52);
//get Stream object
Stream<Integer> openStream = arList.stream();
//Configure stream by filtering out required values
Stream<Integer> fileterdStream = openStream.filter(i -> i >= 20);
long streamCount = fileterdStream.count();
System.out.println(streamCount);
System.out.println("Now in single line");
long newFileterdListCount = arList.stream().filter(i -> i >= 20).count();
System.out.println(newFileterdListCount);
}
}

Q) How to Process elements using sorted()?

- If we want to sort elements inside a stream use this sorted() method.
- We can sort based on default natural sorting order
- If we want to sort using customized sorting order then use comparator.

- Demo

```
        List<Integer> arList =new ArrayList<Integer>();
        arList.add(15);
        arList.add(250);
        arList.add(52);


        //get Stream object
        Stream<Integer> openStream = arList.stream();
        //Configure stream by filtering out required values
        Stream<Integer> fileterdStream = openStream.filter(i -> i >= 20);
        //u can sort on any stream filter or mapped or on normal stream (non filter or non mapped too)
        Stream<Integer> sortedStream  = fileterdStream.sorted();
        sortedStream.forEach(x -> System.out.println(x));

        System.out.println("Now in single line");

        Stream<Integer> newFileterdSortedList = arList.stream().filter(i -> i >= 20).sorted();
        newFileterdSortedList.forEach(x -> System.out.println(x));
    }


}
```

Q) How to Process elements using Min , Max

- Min(Comparator) will return the minimum value based on the defined comparator
- Max(Comparator) will return the maximum value based on the defined comparator

```
14          arList.add(15);
15          arList.add(250);
16          arList.add(52);
17
18
19          //get Stream object
20          Stream<Integer> openStream = arList.stream();
21          //Configure stream by filtering out required values
22          Stream<Integer> fileterdStream = openStream.filter(i -> i >= 20);
23          //u can sort on any stream filter or mapped or on normal stream (non filter or non mapped t
24          Integer min  = fileterdStream.min((i1, i2) -> i1.compareTo(i2)).get();
25          System.out.println(min);
26  //      Integer max  = fileterdStream.max((i1, i2) -> i1.compareTo(i2)).get();
27  //      System.out.println(max);
28
29          System.out.println("Now in single line");
30
31          Integer minValue = arList.stream().min((i1, i2) -> i1.compareTo(i2)).get();
32          System.out.println(minValue);
33          Integer maxValue = arList.stream().max((i1, i2) -> i1.compareTo(i2)).get();
34          System.out.println(maxValue);
35      }
36
```

Q) How to Process elements using forEach?

- forEach() is a method .
- All methods that we saw till now returned something, like min max value, sorted collection, etc
- This method does not return anything.
- Rather This method takes lambda expression as argument and apply that lambda expression to each element present in that stream.

Q) How to Process elements using toArr

We can use this method to copy elements present in the stream to specified array.

- Java 8 demos/src/streams_demo/ToArrayDemo.java - Spring Tool Suite 4

Project SQL Editor Run Database Window Help

Suggested: Spring Boot In

lectDemo.java | CountDemo.java | SortedDemo.java | SortedDescendingOrderDemo.java | MinMaxDemo.java | ToArrayDemo.java

```
        //get Stream object
        Stream<Integer> openStream = arList.stream();
        //Configure stream by filtering out required values
        Stream<Integer> fileterdStream = openStream.filter(i -> i >= 20);
        //u can sort on any stream filter or mapped or on normal stream (non filter or non mapped too)
        Object[] intArr  = fileterdStream.toArray();
        for(Object o : intArr) {
            System.out.println("element in array is " + o);
        }

        System.out.println("Now in single line");

        Object[] intArrOneLiner = arList.stream().filter(i -> i >= 20).toArray();
        for(Object o : intArrOneLiner) {
            System.out.println("element in array is " + o);
        }
    }


}
```

oblems Javadoc Declaration Console Progress

ninated> MinMaxDemo [Java Application] F:\Software\spring-tool-suite-4-4.9.0.RELEASE-e4.18.0-win32.win32.x86_64.self-extracting\contents\sts-4.9.0.RELEA

in single line

Q) How to Process elements using of()?

- Stream concept is not applicable just for the collections it's also applicable for **"ANY GROUP OF VALUE"**.
- Even for arrays you can use stream.
- Stream . Of() this method can take any group of values and convert them to stream so that multiple stream operations can be applied to it.
- Demo

CollectDemo.java | CountDemo.java | SortedDemo.java | SortedDescendingOrderDemo.java | MinMaxDemo.java | ToArr

```
1  package streams_demo;
2
3  import java.util.stream.Stream;
4
5  public class OfDemo {
6
7      public static void main(String[] args) {
8
9          Stream.of(1,11,111,1111,11111).forEach(x -> System.out.println(x));
10
11         String[] name = {"Code", "Decode", "Demos"};
12         Stream.of(name).filter(x -> x.length() > 4).forEach(x -> System.out.println(x));
13
14         }
15
16
17 }
18
```

What is a Parallel Stream?

- Java Parallel Streams came into picture after java 1.8.
- Its meant to utilize multiple cores of processor.
- Till Now our java code has 1 stream of processing where it executes sequentially.
- But when you use parallel streams, we divide code into multiple streams that executes parallely , on separate cores and final result is the outcome of individual cores outcomes combined.

Sequential Stream?

Tasks	Core	Task 1	Task 2	Task 3	Task 4
T1	CORE 1	T1	T2	T3	T4
T2	CORE 2				
T3	CORE 3				
T4	CORE 4				

- The output of this sequential stream is T1 , T2, T3 , T4 → in sequential order tasks are executed and output of 1 can be input to another.

Parallel Stream?

Tasks
T1
T2
T3
T4

Core	Task 1	Task 2	Task 3	Task 4
CORE 1		T2		
CORE 2				T4
CORE 3	T1			
CORE 4			T3	

Tasks	Core	Task 1	Task 2	Task 3	Task 4
T1	CORE 1		T2		
T2	CORE 2				T4
T3	CORE 3	T1			
T4	CORE 4			T3	

The output of this Parallel stream is T2 , T4, T1 , T3 → not in sequential order.

Order of execution is not under control.

Hence its advisable to use parallel stream only when order of execution of threads does not matter and state of one element does not affect another