The addition of the Stream API was one of the major features added to Java 8. A Stream in Java can be defined as a sequence of elements from a source that supports aggregate operations on them. The source here refers to collections or arrays that provide data to a stream.

A few important points about streams are:

- 1. A stream is not a data structure itself. It is a bunch of operations applied to a source. The source can be collections, arrays or I/O channels.
- 2. Streams don't change the original data structure.
- 3. There can be zero or more intermediate operations that transform a stream into another stream.
- 4. Each intermediate operation is lazily executed (This will be discussed later).
- 5. Terminal operations produce the result of the stream.

Stream creation !

Streams can be created from different element sources, e.g., a collection or an array with the help of stream() and of() methods. Below are the different ways to create a stream.

```
a) Stream.of(v1, v2, v3....)
```

In the below example, we are creating a stream of integers at **line** 7 using the **Stream.of()** method.

```
import java.util.stream.Stream;
                                                                                                          C
   public class StreamDemo {
       public static void main(String[] args)
            Stream<Integer> stream = Stream.of(1,2,3,4,5,6,7,8,9);
           stream.forEach(p -> System.out.println(p));
9
10 }
11
Run
                                                                                                  Reset
```

b) List.stream()

In the below example, we are creating a stream from a List at line 14.

```
import java.util.ArrayList;
                                                                                                            G
    import java.util.List;
    import java.util.stream.Stream;
    public class StreamDemo {
 6
        public static void main(String[] args) {
            List<String> list = new ArrayList<>();
 8
            list.add("a");
            list.add("b");
10
11
            list.add("c");
            list.add("d");
12
13
            Stream<String> stream = list.stream();
14
            stream.forEach(p -> System.out.println(p));
15
16
17
18
                                                                                                    Reset
Run
```

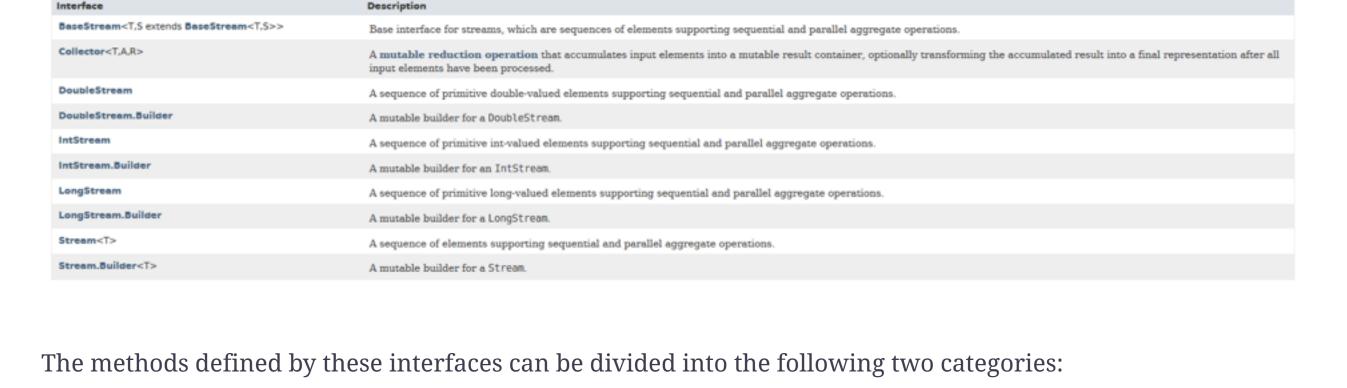
The Stream API defines a few interfaces such as Stream, IntStream, LongStream, etc.

The Stream interfaces

The Stream<T> interface is for object elements. For primitives, it defines IntStream, LongStream and

DoubleStream interfaces. It is a good practice to use primitive streams if you are dealing with primitives because wrapping primitives

to objects and auto-boxing is a costly process. Below is the complete list of methods defined in Stream API.



Intermediate operations

These methods do not produce any results. They usually accept functional interfaces as parameters and

always return a new stream. Some examples of intermediate operations are filter(), map(), etc.

Terminal operations

The streams operations can be further classified as:

These methods produce some results, e.g., count(), toArray(..), and collect(..).

1. filtering

2. slicing

Interface Summary

Interface

- 3. mapping
- 4. matching and finding 5. reduction
- 6. collect

This was the basic introduction to streams. In the next few lessons, we will explore each of these operations. We will also look at how these methods are combined together to process collections.