# JAVA 8

## Sequential vs Parallel Streams

@techy.tacos

*By default*, any stream operation in Java is processed *sequentially*, unless explicitly specified as parallel. Sequential streams *use a single thread* to process the pipeline.

**Example :** The output of below code snippet would be predictable. The list element will always be printed in ordered sequence.

```
public class DemoStreams {

    public static void main(String[] args) {
        //Array list
        List<Integer> listOfNumbers = Arrays.asList(10, 20, 30, 40);
        //sequential stream
        listOfNumbers.stream().forEach(number ->
                System.out.println(number + " " + Thread.currentThread().getName())
        );
    }
}

Output :
10 main
20 main
30 main
40 main
```

Any stream in Java can easily be transformed from sequential to parallel. This can be achieved by adding the parallel method to a sequential stream or by creating a stream using the parallelStream method of a collection. Parallel streams enable us to execute code in parallel on separate cores.

**Example :** The order of execution is out of our control. It may change every time we run the program.

```
public class DemoStreams {

    public static void main(String[] args) {
        List<Integer> listOfNumbers = Arrays.asList(10, 20, 30, 40);
        //parallel stream
        listOfNumbers.parallelStream().forEach(number ->
                System.out.println(number + " " + Thread.currentThread().getName())
        );
    }
}

Output :
40 ForkJoinPool.commonPool-worker-5
20 ForkJoinPool.commonPool-worker-3
10 ForkJoinPool.commonPool-worker-9
30 main
```