### **What Are Streams?**

**Answer: -** Streams can be defined as a sequence of elements from a source which support data processing operations. You can treat streams as operations on data.

**Why Streams?**

**Answer: -** Almost every Java application use Collections API to store and process the data. Despite being the most used Java API, it is not easy to write the code for even some common data processing operations like filtering, finding, matching, sorting, mapping etc. using Collections API. So, there needed Next-Gen API to process the data. So, Java API designers have come with Java 8 Streams API to write more complex data processing operations with much of ease.

### **Characteristics Of Java 8 Streams**

**Answer: -**

#### **Streams are not the data structures: -** Streams doesn’t store the data. You can’t add or remove elements from streams. Hence, they are not the data structures. They are the just operations on data.

#### **Stream Consumes a data source: -** Stream consumes a source, performs operations on it and produces the result. Source may be a collection or an array or an I/O resource. Remember, stream doesn’t modify the source.

#### **Intermediate And Terminal Operations: -** Most of the stream operations return another new stream and they can be chained together to form a pipeline of operations.

The operations which return stream themselves are called intermediate operations. For example – *filter ()*, *distinct ()*, *sorted ()* etc. The operations which return other than stream is called terminal operations. *count ()*. mi*n ()*, *max ()* are some terminal operations.

#### **Pipeline Of Operations: -** A pipeline of operations consists of three things –

#### a source,

#### one or more intermediate operations and

#### a terminal operation.

#### Pipe-lining of operations let you to write database-like queries on a data source. In the below example, int array is the source, *filter ()* and *distinct ()* are intermediate operations and *forEach ()* is a terminal operation.

|  |
| --- |
| IntStream.of(new int[] {4, 7, 1, 8, 3, 9, 7}).filter((int i) -> i > 5).distinct().forEach(System.out::println); |

#### **Internal Iteration: -** Collections need to be iterated explicitly. i.e., you have to write the code to iterate over collections. But all stream operations do the iteration internally behind the scene for you. You need not to worry about iteration at all while writing the code using Java 8 Streams API.

#### **Parallel Execution: -** To gain the performance while processing the large amount of data, you have to process it in parallel and use multi core architectures. Java 8 Streams can be processed in parallel without writing any multi-threaded code.

#### **Streams are lazily populated: -** All elements of a stream are not populated at a time. They are lazily populated as per demand because intermediate operations are not evaluated until terminal operation is invoked.

#### **Streams are traversable only once: -** You can’t traverse the streams more than once just like iterators. If you traverse the stream first time, it is said to be consumed.

#### **Short Circuiting Operations: -** Short circuiting operations are the operations which don’t need the whole stream to be processed to produce a result. For example – *findFirst()*, *findAny()*, *limit()* etc.