**Java 8 Stream API**

Java Stream is a new concept added into Java 8 version that allows us to perform functional-style operations on streams of elements, such as map-reduce transformations on collections.

Java added a new package **java.util.stream** that consists of several classes, interfaces to perform the stream-based operations.

A Java Stream is a component that is capable to perform internal operations of its elements. For example, it can iterate its elements itself.

**Stream operations** are divided into intermediate and terminal operations. A stream source can be a Collection, an array, a generator function, or an I/O channel.

**The intermediate operations** are lazy operations and return a new stream. For example, executing an intermediate operation such as filter() does not actually perform any filtering, but instead creates a new stream.

**The terminal operations** terminate the stream source pipeline after performing the operations. For example, Stream.forEach or IntStream.sum, may traverse the stream to produce a result.

Features Of Stream

**Functional in nature**

A stream is functional in nature that allows us to perform functional-style operations. An operation on a stream produces a result but does not modify its source. For example, filtering a Stream obtained from a collection produces a new Stream without the filtered elements, rather than removing elements from the source collection.

**Laziness-seeking**

Stream uses lazy loading to improve performance. We can use lazy loading in Many stream operations, such as filtering, mapping, or duplicate removal, etc. Stream Intermediate operations are always lazy.

**No storage**

A stream is not a data structure that stores elements, but it conveys elements from a source such as a collection, an array, or a generator function, through a pipeline of computational operations.

**Possibly unbounded**

While Java collections have a finite size, streams need not. Short-circuiting operations such as limit(n) or findFirst() can allow computations on infinite streams to complete in finite time.

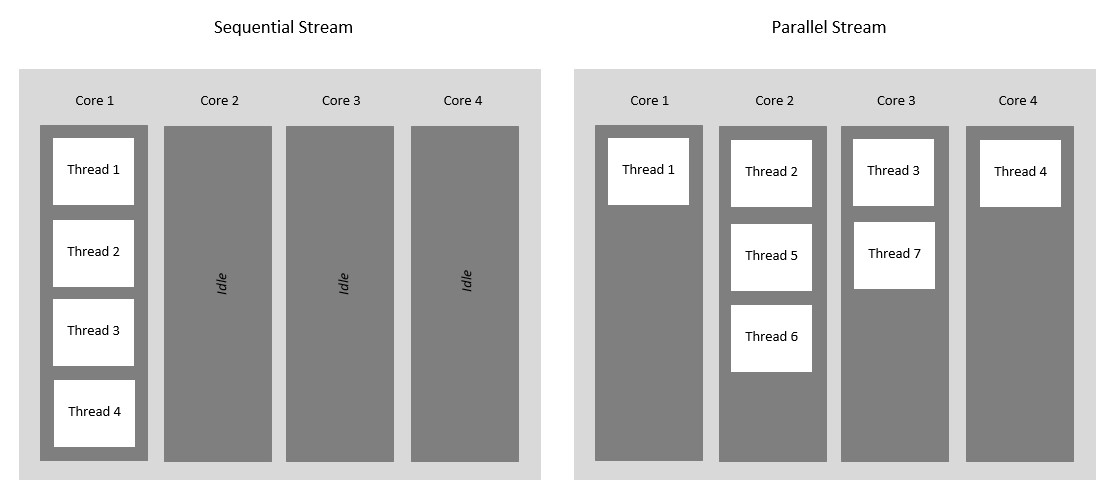
**Consumable**

The elements of a stream are only visited once during the life of a stream. Reusing the stream reference again does not work. We need to create a new stream to revisit the same elements of the source.

# What is Java Parallel Streams?

Java Parallel Streams is a feature of Java 8 and higher, meant for utilizing multiple cores of the processor. Normally any java code has one stream of processing, where it is executed sequentially. Whereas by using parallel streams, we can divide the code into multiple streams that are executed in parallel on separate cores and the final result is the combination of the individual outcomes. The order of execution, however, is not under our control.

Therefore, it is advisable to use parallel streams in cases where no matter what is the order of execution, the result is unaffected and the state of one element does not affect the other as well as the source of the data also remains unaffected.



### Why Parallel Streams?

Parallel Streams were introduced to increase the performance of a program, but opting for parallel streams isn’t always the best choice. There are certain instances in which we need the code to be executed in a certain order and in these cases, we better use sequential streams to perform our task at the cost of performance. The performance difference between the two kinds of streams is only of concern for large-scale programs or complex projects. For small-scale programs, it may not even be noticeable. Basically, you should consider using Parallel Streams when the sequential stream behaves poorly.