**Comprehensive Java Streams: Operations, Pipeline, and Sources**

**JAVA Streams**

Oracle’s Java documentation describes a stream as “A sequence of elements supporting sequential and parallel aggregate operations.” Another way of putting that is, a stream is a set of computational steps against a data set, that are chained together. Streams are a mechanism for describing a whole series of processes, before actually executing them.

**A Stream is different from a Collection**

The stream and the collection types were designed for different purposes. A collection is used to store and manage a series of elements in Java, providing direct access to the Collection elements. We can use collections to manipulate or query a set of data.

A stream was designed to manage the processing of elements. Streams don’t actually store elements, instead these elements are computed on demand, from a data providing source. This collection could be a database result.

***The stream interface is generic, and takes a collection of T, so a stream can process any type that not a primitive. Java provides special Streams for the primitive type as well, with IntStream, DoubleStream and LongStream.***

**The Lazy Stream**

Streams are lazy, like lambda expressions variables. When you can many of the methods on a stream, execution may not immediately occur. Instead, you will need to invoke a special operation on the stream, like you would by calling a lambda’s functional method.

This special operation is called a ***terminal operation.***

**Why Use a Stream?**

Streams are an exciting addition to java, because they provide several benefits.

* First, they make the code to process data uniform, concise and repeatable, in ways that feel similar to a database’s structured query language.
* Second, when working with large collections, parallel stream will provide a performance advantage.