Java Streams - What they are and what they aren't

When you think of a stream, you might be thinking of I/O streams, like a buffered input stream or file output.

That isn't the type of stream I'm talking about here.

Oracle's Java documentation describes a stream as

Stream
A sequence of elements supporting sequential and parallel aggregate operations.

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.

You can use collections to manipulate or query a set of data.

There's nothing you can do with a stream, that you couldn't already do with a Collection.

However, 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.

The Lazy Stream

Another important difference is that streams are lazy, like lambda expression variables.

When you call many of the methods on a stream, execution may not immediately occur.

Instead, you'll 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 (SQL).
- Second, when working with large collections, parallel streams will provide a performance advantage.

All of the code samples I've provided up to this point, using collections, will continue to be valuable for many use cases.

It's time to talk about this new way of doing things, using this additional functional programming feature.