Streams:

Terminal Operators:

They can traverse a stream to produce a result or side effect.

Terminal operations in Java streams are like the tap at the end of a pipeline that produces the desired final output.

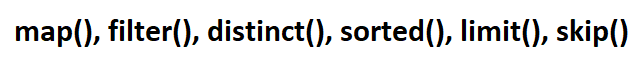
Terminal operations are eager, meaning they complete their traversal of the data source and processing of the pipeline before returning. After a terminal operation is performed, the stream pipeline is considered consumed and can no longer be used.

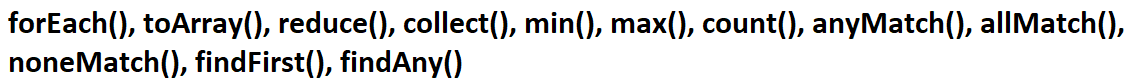
Ex: toList(), forEach(), reduce(), collect, min, max, count, anyMatch, noneMatch, findFirst, findAny, and toArray.

Note: Intermediate operations, such as map and filter, perform some logic and transform stream elements. Intermediate operations are not executed until a terminal operation is called because Java streams use lazy evaluation. Lazy evaluation means that the elements in the stream are only evaluated when necessary, which allows for more efficient processing of large collections of objects.

**Intermediate VS Terminal Operations:**

1. The main difference between intermediate and terminal operations is that intermediate operations return a stream as a result and terminal operations return non-stream values like primitive or object or collection or may not return anything.
2. As intermediate operations return another stream as a result, they can be chained together to form a pipeline of operations. Terminal operations cannot be chained together.
3. Pipeline of operations may contain any number of intermediate operations, but there has to be only one terminal operation, that too at the end of pipeline.
4. Intermediate operations are lazily loaded. When you call intermediate operations, they are not actually executed. They are just stored in the memory and executed when the terminal operation is called on the stream.
5. As the names suggest, intermediate operations don’t give end result. They just transform one stream to another stream. On the other hand, terminal operations give end result.

Intermediate Operations: 

Terminal Operations: 

How can **Comparator** be a Functional Interface when it has **two** abstract methods?

If an interface declares an abstract method **overriding one of the public methods of**java.lang.Object, that **does***not***count** toward the interface's abstract method. since any implementation of the interface will have an implementation from java.lang.Object or elsewhere.

Refer FunctionalInterface.java class documentation.