Stream Intermediate and Terminal Operations

Previously we discussed two parts of Stream Introduction. [Part 1](http://data-structure-learning.blogspot.com/2016/02/stream-introduction.html) and [Part 2](http://data-structure-learning.blogspot.com/2016/03/stream-introduction-part-2.html) we learned streams in general and that streams are declarative type of programming.

After that we saw getting started with Stream<T>. We saw what Stream is? To iterate it Stream is a sequence of elements from a source that supports data processing operations.

Streams are also called pipeline or Stream pipeline.

In this post, we will see what are Intermediate and terminal operations in Stream<T> interface. After this post we will walk through all the methods of Stream<T> interface with practical examples.

Let us start understanding Intermediate Operations.

Intermediate Operations returns or produces a Stream<T>.

Intermediate Operations: Intermediate operations are those that return or produces a Steam<T>. For example, filter() method. The advantage of it is we can connect multiple intermediate operations one after another. The big advantage is that it does not do any computations or processing. It will do processing once it encounters the terminal operations. So Stream’s are lazily evaluated.

Let us take example from previous article.

Step 1: transactions.stream()

Step 2: .filter(transaction -> transaction.getValue() < 100)

Step 3: .map(transaction -> transaction.getPerson().getCity())

Step 4: .distinct()

Step 5: .collect(Collectors.*toList*());

Step 1: Get a Stream of transactions.

Step 2: Filter out all the transactions whose value is less than 100, returns a Stream.

Step 3: Get all the city names of person whose transaction value is greater than 100, returns a Stream.

Step 4: Get all the distinct city names of person whose transaction value is greater than 100, returns a Stream.

Step 5: Collect all the cities to List, returns a List.

filter(), map() and distinct() are intermediate operations as they return a Stream<T>.

collect() method does not return Stream<T>

Terminal Operations:

Terminal Operations returns or produces a non –stream result.

Terminal operations are those that do not return Stream<T> but returns something else i.e. void, boolean, long, Optional<T>, etc. The terminal operations will trigger a computation call. As soon as terminal operation is called the computations starts in pipeline. So the computations are lazy.

We will see few methods in this article. From next article it will be totally hands on.

Few methods of Stream<T> interface, type of operation & its return type.

|  |  |  |
| --- | --- | --- |
| Method name | Type of operation | Return type |
| concat | Intermediate operation | Stream<T> |
| distinct | Intermediate operation | Stream<T> |
| empty | Intermediate operation | Stream<T> |
| filter | Intermediate operation | Stream<T> |
| generate | Intermediate operation | Stream<T> |
| iterate | Intermediate operation | Stream<T> |
| map | Intermediate operation | Stream<R> |
| peek | Intermediate operation | Stream<T> |
| allMatch | Terminal operation | boolean |
| anyMatch | Terminal operation | boolean |
| noneMatch | Terminal operation | boolean |
| collect | Terminal operation | R |
| findAny | Terminal operation | Optional<T> |
| findFirst | Terminal operation | Optional<T> |
| min | Terminal operation | Optional<T> |
| max | Terminal operation | Optional<T> |

That’s all on intermediate and terminal operations. From next article we will work with individual methods of Stream<T> interface with practical examples.

Once again, intermediate operations are those that produces or returns a Stream<T> whereas terminal operations are those that returns non-stream i.e. void, long, Optional<T>, etc.

Terminal Operations returns or produces a non –stream result.

Intermediate Operations returns or produces a Stream<T>.