STREAM IN JAVA

Introduced in Java 8, Stream API is used to process collections of objects. A stream in Java is a sequence of objects that supports various methods which can be pipelined to produce the desired result.

In-short: Stream API is basically perform bulk operations and process the objects of collection.

Use of Stream in Java:

1. Stream API is a way to express and process collections of objects.
2. Enable us to perform operations like filtering, mapping,reducing and sorting.

Create Java Stream:

a) Create stream object using collection object

b) Create stream object using empty(), builder() etc.

Example- Stream<Object> empty = Stream.*empty*();

Java Stream Features:

a) A stream is not a data structure instead it takes input from the Collections, Arrays or I/O channels.

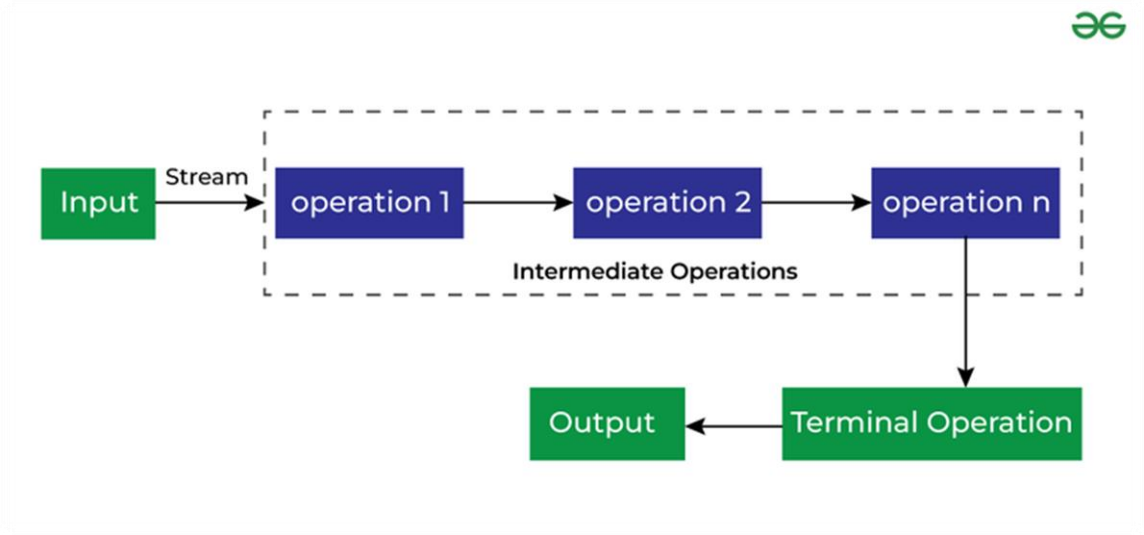
b) Streams don't change the original data structure, they only provide the result as per the pipelined methods.

c) Each intermediate operation is lazily executed and returns a stream as a result, hence various intermediate operations can be pipelined. Terminal operations mark the end of the stream and return the result.

Types of Operations On Streams:

a) Intermediate Operations

b) Terminate Operations



Characteristics of Intermediate Operations:

a) Methods are chained together.

b) Intermediate operations transform a stream into another stream.

c) It enables the concept of filtering where one method filters data and passes it to another method after processing.

Benefit of Java Stream:

a) No Storage

b) Pipeline of Functions

c) Laziness

d) Can be infinite

e) Can be parallelized

f) Can be created from collections, arrays, Files Lines, Methods in Stream, IntStream etc.

Various Intermediate Operations used in Development:

1. map() : The map method is used to return a stream consisting of the results of applying the given function to the elements of this stream.

```
List number = Arrays.asList(2,3,4,5);
List square = number.stream().map(x->x*x).collect(Collectors.toList());
```

2. filter() : The filter method is used to select elements as per the Predicate passed as an argument..

```
List names = Arrays.asList("Reflection","Collection","Stream");
List result =
 names.stream().filter(s->s.startsWith("S")).collect(Collectors.toList
());
```

3. sorted() : The sorted method is used to sort the stream..

```
List names = Arrays.asList("Reflection","Collection","Stream");
List result = names.stream().sorted().collect(Collectors.toList());
```

Various Terminal Operations used in Development:

1. collect(): The collect method is used to return the result of the intermediate operations performed on the stream.

```
List number = Arrays.asList(2,3,4,5,3);
Set square = number.stream().map(x->x*x).collect(Collectors.toSet());
```

2. forEach(): The forEach method is used to iterate through every element of the stream.

```
List number = Arrays.asList(2,3,4,5);
number.stream().map(x->x*x).forEach(y->System.out.println(y));
number.stream().map(x->x*x).forEach(System.out::println);
```

Note: Intermediate Operations are running based on the concept of Lazy Evaluation, which ensures that every method returns a fixed value(Terminal operation) before moving to the next method.

More Examples:

List numbers= Arrays.asList(1,5,3,10,15);

i.
```
Integer minValue = asList.stream().min((x, y) ->
x.compareTo(y)).get();

System.out.println("Min Value- " + minValue);
```

ii.
```
Integer maxValue = asList.stream().max((x, y) ->
x.compareTo(y)).get();

System.out.println("Max Value- " + maxValue);
```

Reference: geeksforgeeks