**Streams in Java**

A stream is **a sequence of objects that supports various methods which can be pipelined to produce the desired result**.

The features of the Java stream are –

* A stream is not a data structure instead it takes input from the Collections, Arrays or I/O channels.
* Streams don’t change the original data structure, they only provide the result as per the pipelined methods.
* 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.

Different Operations on Streams-

**Intermediate Operations:**

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());

1. **filter**: The filter method is used to select elements as per the Predicate passed as an argument. It returns a new stream consisting of the elements of the stream from which it is called which are according to the predicate (condition).

List names = Arrays.asList("Reflection","Collection","Stream");

List result = names.stream().filter(s->s.startsWith("S")).collect(Collectors.toList());

1. **sorted**: The sorted method is used to sort the stream. Returns a stream consisting of the elements of the stream passed, sorted according to the natural order.  If the elements of this stream are not comparable, a  java.lang.ClassCastException may be thrown when the terminal operation is executed.

List names = Arrays.asList("Reflection","Collection","Stream");

List result = names.stream().sorted().collect(Collectors.toList());

1. **distinct:** It returns a stream consisting of the distinct(different) elements of the passed stream. For ordered stream, the selection of the distinct elements is stable (For duplicated elements, the element appearing first in the encounter order is preserved). While for non-ordered streams it does not make any guarantee for stability.
2. // To find distinct elements from the list.
3. intList.stream().distinct().sorted().forEach(
4. element -> System.out.print(element + " "));

**Other intermediate operations are limit(), skip()**

***Note:***

* ***Intermediate functions return a stream back.***
* ***On any stream you can execute any number of intermediate operations, but the terminal operation should be single and written at last. So following are the intermediate methods provided by the Stream***
* ***Predicate is a non-interfering, stateless predicate to apply to each element to determine if it should be included or not.***

**Terminal Operations:**

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());

1. **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));

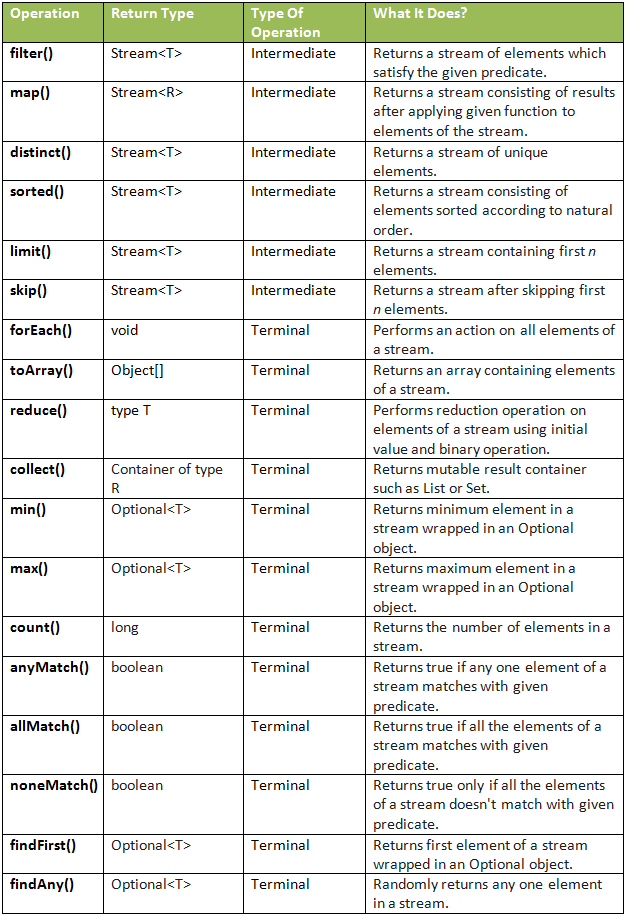
1. **reduce:** The reduce method is used to reduce the elements of a stream to a single value.  
   The reduce method takes a BinaryOperator as a parameter.

List number = Arrays.asList(2,3,4,5);  
int even = number.stream().filter(x->x%2==0).reduce(0,(ans,i)-> ans+i);

There are many other termination operations

**Terminal Operations :**

toArray(), min(), max(), count(), anyMatch(), allMatch(), noneMatch(), findFirst(), findAny()



Example of Streams

//a simple program to demonstrate the use of stream in java

import java.util.\*;

import java.util.stream.\*;

class Demo

{

  public static void main(String args[])

  {

    // create a list of integers

    List<Integer> number = Arrays.asList(2,3,4,5);

    // demonstration of map method

    List<Integer> square = number.stream().map(x -> x\*x).

                           collect(Collectors.toList());

    System.out.println(square);

    // create a list of String

    List<String> names =

                Arrays.asList("Reflection","Collection","Stream");

    // demonstration of filter method

    List<String> result = names.stream().filter(s->s.startsWith("S")).

                          collect(Collectors.toList());

    System.out.println(result);

    // demonstration of sorted method

    List<String> show =

            names.stream().sorted().collect(Collectors.toList());

    System.out.println(show);

    // create a list of integers

    List<Integer> numbers = Arrays.asList(2,3,4,5,2);

    // collect method returns a set

    Set<Integer> squareSet =

         numbers.stream().map(x->x\*x).collect(Collectors.toSet());

    System.out.println(squareSet);

    // demonstration of forEach method

    number.stream().map(x->x\*x).forEach(y->System.out.println(y));

    // demonstration of reduce method

    int even =

       number.stream().filter(x->x%2==0).reduce(0,(ans,i)-> ans+i);

    System.out.println(even);

  }

}

[4, 9, 16, 25]

[Stream]

[Collection, Reflection, Stream]

[16, 4, 9, 25]

4

9

16

25

6

**Important Points/Observations:**

1. A stream consists of source followed by zero or more intermediate methods combined together (pipelined) and a terminal method to process the objects obtained from the source as per the methods described.
2. Stream is used to compute elements as per the pipelined methods without altering the original value of the object.