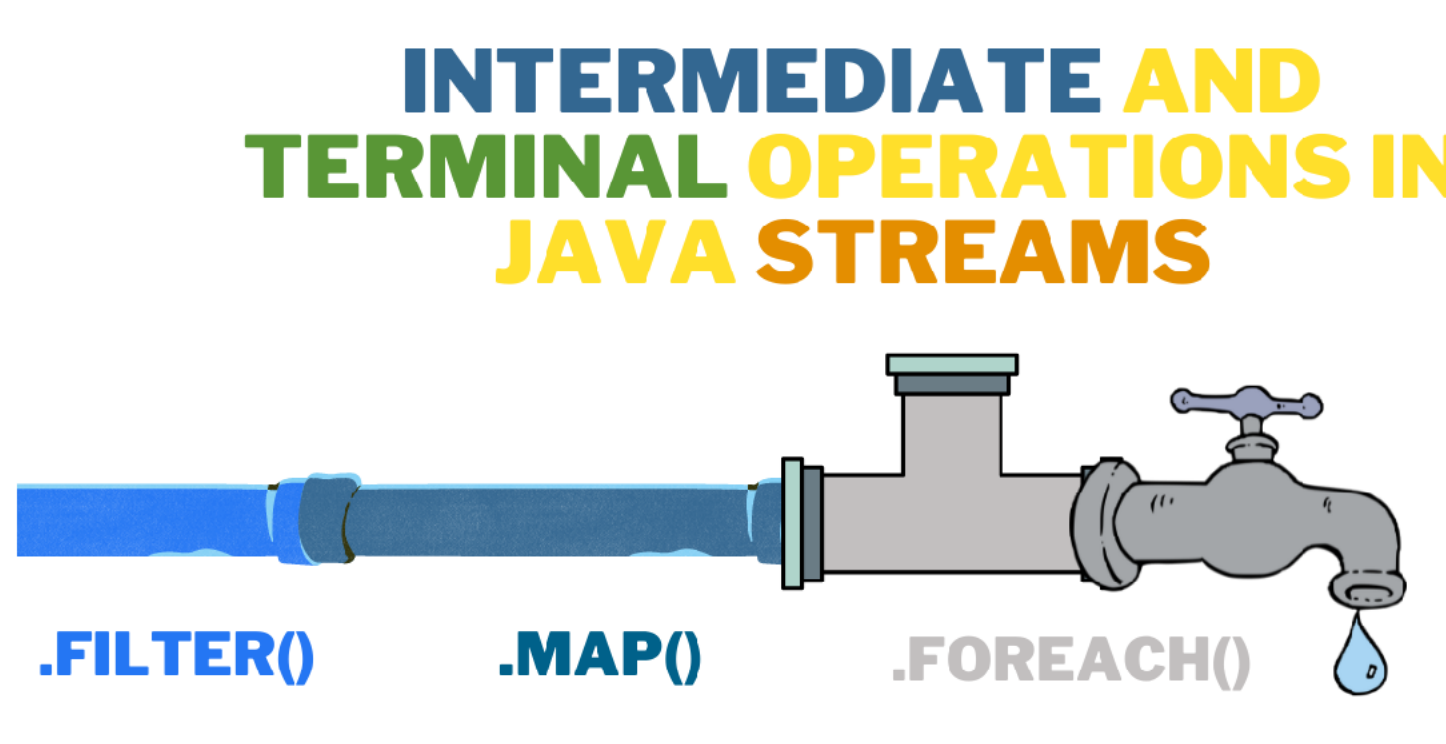
Operations on Stream API :
----
* On Stream API object, We can perform two types of operation

1) Intermediate Operation
2) Terminal Operation

Intermediate Operation :
----
* An intermediate operation methods are not producing a final result, actually they are producing another stream which is known as Intermediate operation.

* It represents a pipleline operation so our stream data will pass through this pipeline and meanwhile we can perform some operation.

* As we know, intermediate operation methods are producing another Stream so all the methods which belongs to intermediate operation will provide java.util.stream.Stream interface as a return type so method chaining is possible.



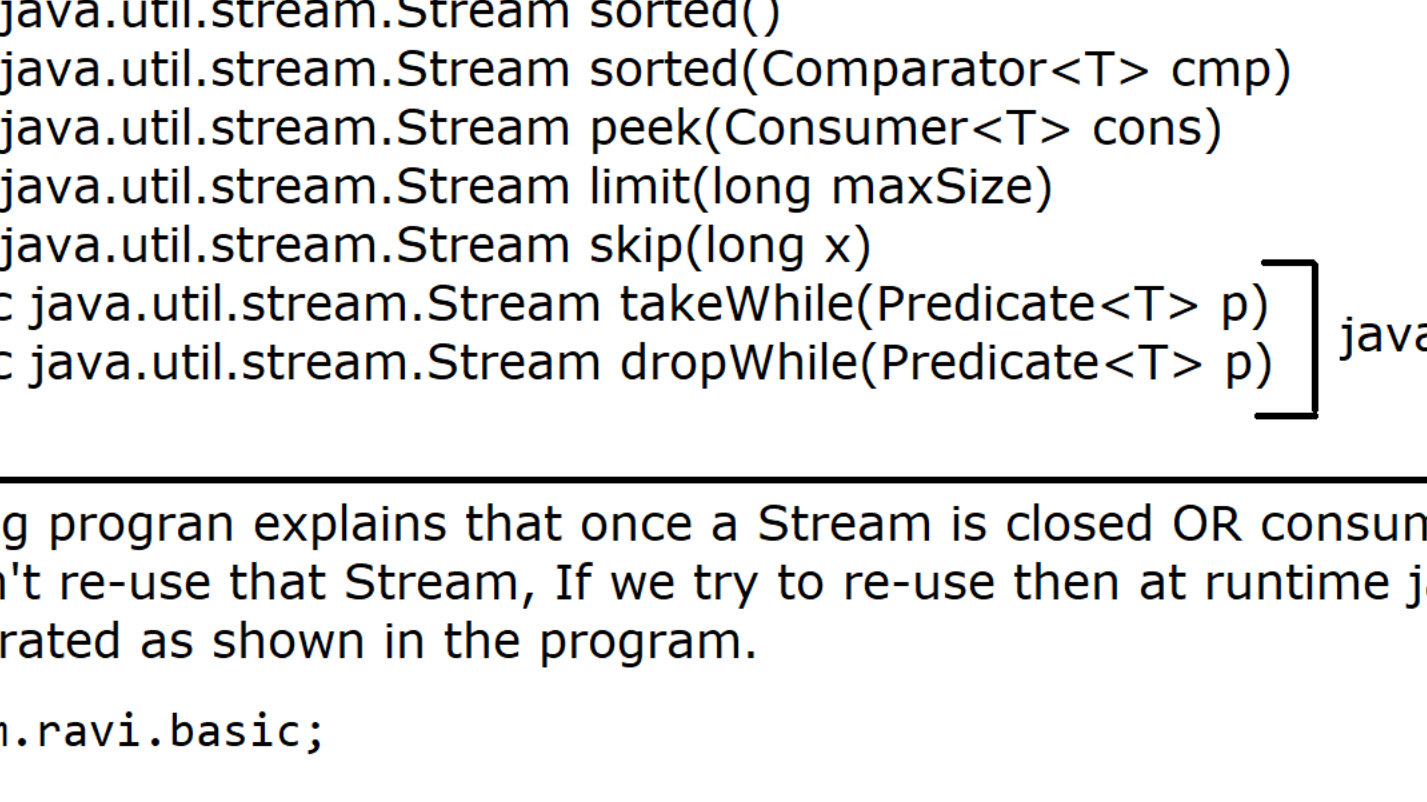
* It provides **lazy evaluation** that means the computation will not be performed till we invoked any terminal method.

* In order to get the result we should compulsory use any of the terminal method, If we use only intermediate methods then result will not come.

* Once we apply terminal method on the Stream for output then the Stream is considered as closed OR consumed so further we cannot apply any operation, If we try to do then we will get a runtime exception i.e. **java.lang.IllegalStateException**

* When we apply intermediate operation on stream object then original data will remain unchanged.

Example with Diagram :



* The following are the methods of intermediate operation :

1) public java.util.stream.Stream filter(Predicate<T> p)
2) public java.util.stream.Stream map(Function<T,R> mapper)
3) public java.util.stream.Stream flatMap(Function<T, **Stream** R> mapper)
4) public java.util.stream.Stream distinct()
5) public java.util.stream.Stream sorted()
6) public java.util.stream.Stream sorted(Comparator<T> cmp)
7) public java.util.stream.Stream peek(Consumer<T> cons)
8) public java.util.stream.Stream limit(long maxSize)
9) public java.util.stream.Stream skip(long x)
10) public java.util.stream.Stream takeWhile(Predicate<T> p)
11) public java.util.stream.Stream dropWhile(Predicate<T> p)

(10 and 11: java 9v onwards)

The following progran explains that once a Stream is closed OR consumed by using terminal method then we can't re-use that Stream, If we try to re-use then at runtime java.lang.IllegalStateException will be generated as shown in the program.

```
package com.ravi.basic;

import java.util.stream.Stream;

public class StreamExceptionOnClosed
{
    public static void main(String[] args)
    {
        Stream<Integer> streamOfNum = Stream.of(1,2,3,4,5,6,7,8);
        streamOfNum.forEach(System.out::println);

        System.out.println("..............");

        streamOfNum.forEach(System.out::println);

    }

}
```

---

public abstract Stream<T> filter(Predicate<T> p) :

It is a predfined intermediate method of Stream interface. It is used to select/filter elements as per the Predicate passed as an argument. It is basically used to filter the elements based on boolean condition. It will accept all the elements which will satisfy the given predicate.

public abstract <T> collect(java.util.stream.Collector c)

It is a predfined method of Stream interface. It is used to return the result of the intermediate operations performed on the stream.

It is a terminal operation. It is used to collect the data after filteration and convert the data to the Collection(List/Set/Map).

Collectors is a predfined final utility class available in java.util.stream sub package which conatins static method toList(),toSet(), toMap() to convert the data as a List/Set/Map i.e Collection object. The return type of these methods (toList(). toSet() and toMap()) Collector interface.

```
package com.ravi.filter;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

/*
 * 1) Filter all the even numbers from Collection with and without Stream API.
 */

public class FilterDemo1
{
    public static void main(String[] args)
    {
      List<Integer> listOfNumber = Arrays.asList(1,2,3,4,5,6,7,8,9,10,11,12);

      //Without Stream API
      List<Integer> evenList = new ArrayList<Integer>();

      for(Integer num : listOfNumber)
      {
          if(num%2==0)
          {
              evenList.add(num);
          }
      }
      evenList.forEach(System.out::println);

      System.out.println(".......................");

      //With Stream API

      listOfNumber.stream()
            .filter(num -> num%2==0)
               .forEach(System.out::println);

    }
}
```

```
package com.ravi.filter;

import java.util.List;
import java.util.Set;
import java.util.stream.Collectors;

/*
    2) Retrieve all the names which starts from character A using Stream API and Convert
the Stream to Set interface.
       */

public class FilterDemo2
{
    public static void main(String[] args)
    {
        List<String> listOfName =
List.of("Aryan","Ankit","Raj","Rohit","Aniket","Raj","Aryan","Ajinkya","Ankit");
        System.out.println(listOfName);

        System.out.println("...............");

        Set<String> filteredName = listOfName.stream()
            .filter(str -> str.startsWith("A"))
                .collect(Collectors.toSet());

        System.out.println(filteredName);

    }

}
```

```
package com.ravi.filter;

import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.Stream;

/*
 * 3) Fetch all the Employees name whose salary is greater than 50k
   and convert into List object.
 */

record Employee(Integer id, String name, Double sal)
{

}

public class FilterDemo3
{
    public static void main(String[] args)
    {
       Employee e1 = new Employee(111, "Juber", 90000D);
       Employee e2 = new Employee(222, "Aryan", 40000D);
       Employee e3 = new Employee(333, "Scott", 60000D);
       Employee e4 = new Employee(444, "Rahul", 70000D);
       Employee e5 = new Employee(555, "Aakash",45000D);
       Employee e6 = new Employee(666, "Manav", 92000D);
       Employee e7 = new Employee(111, "Juber", 90000D);


       List<Employee> filteredEmployee = Stream.of(e1,e2,e3,e4,e5,e6,e7)
           .filter(emp -> emp.sal()>50000)
                 .collect(Collectors.toList());

       filteredEmployee.forEach(System.out::println);

    }
}
```

---

public Object[] toArray() :
----
* It is a predefined method of Stream interaface.
* It is used to convert the Stream into object array.

```
package com.ravi.filter;

/*
 * 4) WAP to print all the names whose length is greater than 3 from String array
      and convert into array
 */

import java.util.Arrays;

public class FilterDemo4
{
    public static void main(String[] args)
    {
        String[] names = {"Scott", "Raj", "Riya", "Smith", "Sachin"};

        Object[] array = Arrays.stream(names)
              .filter(name -> name.length()>3)
                  .toArray();

        System.out.println(Arrays.toString(array));

    }

}
```

```
package com.ravi.filter;

import java.util.Arrays;

//5) WAP to filter all the prime number from the given array.


public class FilterDemo5
{
    public static void main(String[] args)
    {
        int[] numbers = {2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13};

        // Filtering prime numbers
        int[] primeNumbers = Arrays.stream(numbers)
                                   .filter(num -> FilterDemo5.isPrime(num))
                                   .toArray();

        System.out.println("Prime Numbers: " + Arrays.toString(primeNumbers));
    }

    // Method to check if a number is prime
    private static boolean isPrime(int num)
    {
        if (num < 2)
        {
           return false;
        }
        for (int i = 2; i <= Math.sqrt(num); i++)
        {
            if (num % i == 0)
            {
              return false;
            }
        }
        return true;
    }
}
```

//Program on Lazy Evaluation :

```
package com.ravi.filter;

import java.util.Arrays;
import java.util.List;

//Lazy Evaluation

public class FilterDemo6
{

    public static void main(String[] args)
    {
        List<String> fruits = Arrays.asList("Apple", "Banana", "Cherry");

        fruits.stream()
           .filter(fruit->
           {
            System.out.println("Filtering :"+fruit);
            return fruit.startsWith("A");
           }).forEach(System.out::println);
    }

}
```