***Itermediate operations***

* *Intermediate operations transform a stream into another stream*
* *they are lazy, meaning they don't execute until a terminal operation is invoked.*

***filter***

* *Values provided under filter should be predicate method*

List<String> list = Arrays.*asList*("Akshit", "Ram", "Shyam", "Ghanshyam", "Akshit");  
Stream<String> filteredStream = list.stream().filter(x -> x.startsWith("A"));  
// no filtering at this point  
long res = list.stream().filter(x -> x.startsWith("A")).count();  
System.*out*.println(res);

***map***

* *Values provided under map should be functions method*

List<String> list = Arrays.*asList*("Akshit", "Ram", "Shyam", "Ghanshyam", "Akshit");

Stream<String> stringStream = list.stream().map(String::toUpperCase);

***Sorted***

* *Used to sort the stream*

List<String> list = Arrays.*asList*("Akshit", "Ram", "Shyam", "Ghanshyam", "Akshit");

Stream<String> sortedStream = list.stream().sorted();

* *Used to sort the stream based on comparator provided*

List<String> list = Arrays.*asList*("Akshit", "Ram", "Shyam", "Ghanshyam", "Akshit");

Stream<String> sortedStreamUsingComparator = list.stream().sorted((a, b) -> a.length() - b.length());

***Distinct***

* *No duplicated values*

List<String> list = Arrays.*asList*("Akshit", "Ram", "Shyam", "Ghanshyam", "Akshit");

System.*out*.println(list.stream().filter(x -> x.startsWith("A")).distinct().count());

***skip***

* *No duplicated values*

System.*out*.println(Stream.*iterate*(1, x -> x + 1).skip(10).limit(100).count());

***peek*** *.*

* *Values provided under map should be functions method*
* *Performs operation on each element as it is consumed*
* *Majorly used to print element*

Stream.*iterate*(1, x -> x + 1).skip(10).limit(100).peek(System.*out*::println).count()

***flatMap****.*

* *Handles streams of collections, lists, or arrays where each element is itself a collection*
* *Flatten nested structures (e.g., lists within lists) so that they can be processed as a single sequence of elements*
* *Transform and flatten elements at the same time.*
* *Values provided under flatmap should be functions method*

List<List<String>> listOfLists = Arrays.*asList*(  
 Arrays.*asList*("apple", "banana"),  
 Arrays.*asList*("orange", "kiwi"),  
 Arrays.*asList*("pear", "grape")  
 );  
 System.*out*.println(listOfLists.get(1).get(1));  
 System.*out*.println(listOfLists.stream().flatMap(x -> x.stream()).map(String::toUpperCase).toList());  
 List<String> sentences = Arrays.*asList*(  
 "Hello world",  
 "Java streams are powerful",  
 "flatMap is useful"  
 );  
 System.*out*.println(sentences  
 .stream()  
 .flatMap(sentence -> Arrays.*stream*(sentence.split(" ")))  
 .map(String::toUpperCase)  
 .toList());  
  
}

***o/p***

***kiwi***

***[APPLE, BANANA, ORANGE, KIWI, PEAR, GRAPE]***

***[HELLO, WORLD, JAVA, STREAMS, ARE, POWERFUL, FLATMAP, IS, USEFUL]***