***The reduce() operation*** in Java Stream API is a [terminal operation](https://javatechonline.com/stream-api-in-java-8/#Terminal_Operations) that combines the elements of a stream into a single value. It can be used with any type of stream, not just numerical streams. The reduce() operation returns an Optional object that may or may not contain the result, depending on whether the stream was empty or not.

**filter()** is an [intermediate operation](https://javatechonline.com/stream-api-in-java-8/#Intermediate_Operations) in Java Stream API, which is used to filter out elements from a stream based on some condition. It returns a new stream that contains only the elements that satisfy the condition.

**What is a stream in Java 8?**  
A stream is a sequence of elements that can be processed parallel or sequentially. Streams are a new addition to Java 8 and provide an easy way to work with collections.

**What is a method reference in Java 8?**  
A method reference is a shorthand way to write a lambda expression that calls a method. Method references can be used to simplify code and improve readability by eliminating the need for a separate lambda expression.

x->System.out.println(x) 🡪System.out::println

Intermediate Operation:

Operations that are done on a stream and they return stream.

Reduce, filter,Distinct,Sorted

Terminal Operation:

Operation that are done on stream and they return something other than stream.

Either a void or return a value.

Reduce ,collect, forEach

**Functional Interface:**

It has one abstract method.

**Filter** 🡪 x-> x%2==0 🡪 Instance of **Predicate** class is used

Abstract method 🡪 test()

It takes an input returns a boolean

**Map**🡪 x->x\*x 🡪 Instance of **Function** class is used

It takes input returns a value/output (input and output can be different)

Abstract method 🡪 apply ()

**ForEach** ->sout ->Instance of **Consumer** Class is used

It takes input and returns void

Abstract method 🡪 accept ()

Function descriptor 🡪test(),accept(),apply()

Sending code/logic as an argument to a function is called Behaviour Parametrization

**Other functional interface:**

**BinaryOperator** 🡪 takes 2 input and returns an output of same type

(x,y)->x+y ;

**Supplier**🡪 opposite of consumer. It does not have input and produces a output

()->2 ; () ->{return 2; }

eg: random number generation

Max:

Returns the maximum element of this stream according to the provided Comparator. This is a special case of a reduction.

This is a terminal operation.

* **Average/Sum/Count works on mapToInt**

**Grouping:**

**🡪 collect(Collectors.groupingBY(Course::getCategory)**

**🡪 collect(Collectors.groupingBY(Course::getCategory,collectors.counting)**

**IntSteam🡪 collections….Do boxed() before collect**

**Why functional pgm?**

**🡪Performance:**

System.out.println(courses.stream().filter((course>course.length()>5)).map(String::toUpperCase).findFirst());

Streams executes all intermediate operation only after terminal operations is

executed and hence they are Lazy.

performance is good as it stops execution as soon as conditions are met

in structured programming, state of the object is affected during the process of executing the function.

But in stream, we do the operations in streams instead of the actual object.

Hence parallel() method divides the job it to mini jobs and unite te results which reduces time and hence performance is good

Which of the following methods can be used to convert a stream of strings to a list of integers?-->mapToInt**(Integer::parseInt)**

Which of the following methods can be used to convert a stream of integers to a list of strings?-->**map(String::valueOf)**

stream operations is used to generate an infinite stream of elements in Java 8? **--> generate()**

**Read a file:**

Files.lines(Paths.get("file.txt")).forEach(System.out::println);