Filter 🡪 Predicate 🡪one argument returns boolean

Map🡪 Function 🡪one argument returns a value(input,output can be different)

forEach🡪 Consumer 🡪one argument does not return

Integer:: sum 🡪– BinaryOperator –> two input, one output (input,output be of same type)

Supplier : 🡪No input but return value/date

Integer:: sum 🡪– UnaryOperator –> one input, one output (input,output be of same type)

Bi Predicate 🡪 2 iputs of any type and return boolean

BiFunc 🡪 Two argument returns a value

BiConsumer 🡪 2 argument does not return any value

==================================================================================

numbers.stream()

.filter(n->n%2==0)

.map(x->x\*x)

.forEach(System.out::println);

==================================================================================

Int sum =numbers.stream().reduce(0, Integer::sum)

Reduce 🡪

Integer:: sum 🡪functional interface – BinaryOperator

Bifunction 🡪Apply func

=====================================================================================

private static void distinctSortedListFunc(List<Integer> numbers) {

numbers.stream()

.sorted()

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

}

=====================================================================================

private static void distinctRevSortedListFunc(List<Integer> numbers) {

numbers.stream()

.sorted(Comparator.reverseOrder())

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

}

====================================================================================

private static void LenSortedListFunc(List<String> list) {

list.stream()

.sorted(Comparator.comparing(String::length))

.forEach(System.out::println);

}

====================================================================================

Behaviour Parametrization:

Sending a behaviour as a parameter of a method

List<Integer> list = List.of(1,2,3,4);

filterAndPrint(list, x1 -> x1 % 2 == 0);

//filterAndPrint(list, x -> x % 2 == 1);

}

private static void filterAndPrint(List<Integer> list, Predicate<Integer> Predicate) {

list.stream()

.filter(Predicate)

.forEach(System.out::println);

}

============================================================================

Supplier:

Supplier<Integer> randomQ = ()-> {

Random r = new Random();

return r.nextInt(1000);

};

System.out.println(randomQ.get());

===========================================================

UnaryOperator:

UnaryOperator<Integer> uo = x->x\*3;

System.out.println(uo.apply(10));

===========================================================

Method reference 🡪 on various methods

ls.stream()

.map(str-> str.toUpperCase()) 🡪 Method reference on instance method

.forEach(System.out::println);🡪 Method reference on static method

Supplier<String> supp = () -> new String(); 🡪 constructor reference

ls.stream()

.map(String::toUpperCase)

.forEach(BehaviorParametrization::print);

Supplier<String> suppl = String ::new;

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

[ File.txt in main directory ]

Read words from a file

Files.lines(Paths.get("file.txt"))

.map(str->str.split(" "))

.flatMap(Arrays::stream)

.distinct()

.forEach(System.out::println);

Important :

MATCH 🡪 returns boolean - Predicate

allMatch --> if all matching data

noneMatch --> if all unmatched

anyMatch --> if any one matched

------------------------------------------

SORT

Asc order sort -> define compartor and use it in stream with sort

Desc order sort ->define reverse() at the end of compartor and use in stream

Two comparison -> define compartor , thencomparing and use comparator variable in stream

REMOVE DATA

limit (5) -> gives first 5 data

Skip (5) -> gives data after 5th element

takeWhile -> gives first available elements until condition is met

dropwhile--> gives elements that are present after condition is met

--------------------------------------------------------------------------------------------------------