List<String> stringList = **new** ArrayList<>();

stringList.add("Subhajit");

stringList.add("Asim");

stringList.add("Suman");

stringList.add("Ram");

stringList.add("Asim");

stringList.add("Sourav");

System.***out***.println("stringList : " + stringList);

//allMatch(Predicate)

**boolean** allMatch = stringList.stream().allMatch(val -> {

**return** !(val.contains(" "));

});

System.***out***.println("allMatch : " + allMatch);

//anyMatch(Predicate)

**boolean** anyMatch = stringList.stream().anyMatch(val -> {

**return** val.matches("S(.\*)");

});

System.***out***.println("anyMatch : " + anyMatch);

//noneMatch(Predicate)

**boolean** noneMatch = stringList.stream().noneMatch(val -> {

**return** val.matches("Z(.\*)");

});

System.***out***.println("noneMatch : " + noneMatch);

**long** count = stringList.stream().count();

System.***out***.println("count : " + count);

List<String> distinctStringList = stringList.stream().distinct().collect(Collectors.*toList*());

System.***out***.println("distinctStringList : " + distinctStringList);

//filter(Predicate)

List<String> filteredStringList = stringList.stream().filter(val -> {

**return** val.matches("S(.\*)");

}).collect(Collectors.*toList*());

System.***out***.println("filteredStringList : " + filteredStringList);

Optional<String> findAny = stringList.stream().findAny();

System.***out***.println("findAny : " + findAny.get());

Optional<String> findFirst = stringList.stream().findFirst();

System.***out***.println("findFirst : " + findFirst.get());

//forEach(Consumer)

stringList.stream().forEach(val -> {

System.***out***.println("Data by forEach : " + val);

});

//limit(Long)

List<String> limitedStringList = stringList.stream().limit(3).collect(Collectors.*toList*());

System.***out***.println("limitedStringList : " + limitedStringList);

//max(Comparator)

Optional<String> max = stringList.stream().max((val1, val2) ->{

**return** val1.compareTo(val2);

});

System.***out***.println("max : " + max.get());

//min(Comparator)

Optional<String> min = stringList.stream().min((val1, val2) ->{

**return** val1.compareTo(val2);

});

System.***out***.println("min : " + min.get());

//sorted() means Aesc order

List<String> sortedStringListAsc = stringList.stream().sorted().collect(Collectors.*toList*());

System.***out***.println("sortedStringListAsc : " + sortedStringListAsc);

//sorted(Comparator)

List<String> sortedStringListDesc = stringList.stream().sorted(Comparator.*reverseOrder*()).collect(Collectors.*toList*());

System.***out***.println("sortedStringListDesc : " + sortedStringListDesc);

//map(Function)

List<String> dataUppercaseUsingMap = stringList.stream().map(val -> {

**return** val.toUpperCase();

}).collect(Collectors.*toList*());

System.***out***.println("dataUppercaseUsingMap : " + dataUppercaseUsingMap);

//mapToInt(ToIntFunction)

List<Integer> dataLengthUsingMapToInt = **new** ArrayList<>();

stringList.stream().mapToInt(val -> {

**return** val.length();

}).forEach(valInt -> {

dataLengthUsingMapToInt.add(valInt);

});

System.***out***.println("dataLengthUsingMapToInt : " + dataLengthUsingMapToInt);

List<String> subListOne = **new** ArrayList<>();

subListOne.add("Z");

subListOne.add("A");

subListOne.add("Y");

subListOne.add("B");

List<String> subListTwo = **new** ArrayList<>();

subListTwo.add("M");

subListTwo.add("A");

subListTwo.add("N");

subListTwo.add("Y");

List<List<String>> superList = **new** ArrayList<>();

superList.add(subListOne);

superList.add(subListTwo);

System.***out***.println("superList : " + superList);

System.***out***.println("superList Count : " + superList.stream().count());

//flatMap(Function)

List<String> superListCombined = superList.stream().flatMap(val -> {

**return** val.stream();

}).collect(Collectors.*toList*());

System.***out***.println("superListCombined : " + superListCombined);

List<String> superListCombinedLowercase = superList.stream().flatMap(val -> {

**return** val.stream();

}).map(val -> {

**return** val.toLowerCase();

}).collect(Collectors.*toList*());

System.***out***.println("superListCombinedLowercase : " + superListCombinedLowercase);

List<String> superListCombinedTwo = **new** ArrayList<>();

superList.stream().map(val -> {

**return** val.stream();

}).forEach(val -> {

val.forEach(valString -> {

superListCombinedTwo.add(valString);

});

});

System.***out***.println("superListCombinedTwo : " + superListCombinedTwo);

List<Integer> dataLengthUsingFlatMapToInt = **new** ArrayList<>();

superList.stream().flatMapToInt(val -> {

IntStream is = val.stream().mapToInt(valString -> {

**return** valString.length();

});

**return** is;

}).forEach(valInt -> {

dataLengthUsingFlatMapToInt.add(valInt);

});

System.***out***.println("dataLengthUsingFlatMapToInt : " + dataLengthUsingFlatMapToInt);

List<Integer> bigList = **new** ArrayList<>();

**for**(**int** i=0;i<5;i++)

bigList.add(i);

bigList.stream().sequential().forEach(valInt -> {

System.***out***.println("sequential -> " + valInt);

});

bigList.stream().parallel().forEach(valInt -> {

System.***out***.println("parallel forEach -> " + valInt);

});

bigList.stream().parallel().forEachOrdered(valInt -> {

System.***out***.println("parallel forEachOrdered -> " + valInt);

});

//------------------------------------ Stream Methods analysis ends here --------------------------------------------------

//------------------------------------ Collectors Methods analysis starts here --------------------------------------------------

//averagingInt(ToIntFunction)

Double averageOfStringLength = stringList.stream().collect(Collectors.*averagingInt*(val -> {**return** val.length();}));

System.***out***.println("averageOfStringLength : " + averageOfStringLength);

Long countOfStringLength = stringList.stream().collect(Collectors.*counting*());

System.***out***.println("countOfStringLength : " + countOfStringLength);

String joinedString = stringList.stream().collect(Collectors.*joining*());

System.***out***.println("joinedString : " + joinedString);

String joinedStringDelimeter = stringList.stream().collect(Collectors.*joining*(","));

System.***out***.println("joinedStringDelimeter : " + joinedStringDelimeter);

String joinedStringDelimeterSufPref = stringList.stream().collect(Collectors.*joining*(",", "prefix : ", " : suffix"));

System.***out***.println("joinedStringDelimeterSufPref : " + joinedStringDelimeterSufPref);

//toMap(Function, Function)

Map<String, Integer> mapStringLength = stringList.stream().distinct().collect(Collectors.*toMap*(val -> {**return** val;}, val -> {**return** val.length();}));

System.***out***.println("mapStringLength : " + mapStringLength);

Set<String> setString = stringList.stream().distinct().collect(Collectors.*toSet*());

System.***out***.println("setString : " + setString);

//mapping(Function, Collector)

List<Integer> stringLengthList = stringList.stream().collect(Collectors.*mapping*(val -> {**return** val.length();}, Collectors.*toList*()));

System.***out***.println("stringLengthList : " + stringLengthList);

//maxBy(Comparartor)

Optional<String> maxString = stringList.stream().collect(Collectors.*maxBy*((val1, val2) -> {

**return** val1.compareTo(val2);

}));

System.***out***.println("maxString : " + maxString.get());

//minBy(Comparartor)

Optional<String> minString = stringList.stream().collect(Collectors.*minBy*((val1, val2) -> {

**return** val1.compareTo(val2);

}));

System.***out***.println("minString : " + minString.get());

//groupingBy(Function, Collector)

Map<Object, Long> groupByFunctionCollector = stringList.stream().collect(Collectors.*groupingBy*(n -> {**return** n.charAt(0);}, Collectors.*counting*()));

System.***out***.println("groupByFunctionCollector : " + groupByFunctionCollector);

//groupingBy(Function)

Map<Object, List<String>> groupByFunction= stringList.stream().collect(Collectors.*groupingBy*(n -> {**return** n.charAt(0);}));

System.***out***.println("groupByFunction : " + groupByFunction);

//groupingBy(Function, Supplier, Collector)

HashMap<Object, Long> groupByFunctionSupplierCollector = stringList.stream().collect(Collectors.*groupingBy*(n -> {**return** n.charAt(0);}, HashMap::**new**, Collectors.*counting*()));

System.***out***.println("groupByFunctionSupplierCollector : " + groupByFunctionSupplierCollector);

OUTPUT :

stringList : [Subhajit, Asim, Suman, Ram, Asim, Sourav]

allMatch : true

anyMatch : true

noneMatch : true

count : 6

distinctStringList : [Subhajit, Asim, Suman, Ram, Sourav]

filteredStringList : [Subhajit, Suman, Sourav]

findAny : Subhajit

findFirst : Subhajit

Data by forEach : Subhajit

Data by forEach : Asim

Data by forEach : Suman

Data by forEach : Ram

Data by forEach : Asim

Data by forEach : Sourav

limitedStringList : [Subhajit, Asim, Suman]

max : Suman

min : Asim

sortedStringListAsc : [Asim, Asim, Ram, Sourav, Subhajit, Suman]

sortedStringListDesc : [Suman, Subhajit, Sourav, Ram, Asim, Asim]

dataUppercaseUsingMap : [SUBHAJIT, ASIM, SUMAN, RAM, ASIM, SOURAV]

dataLengthUsingMapToInt : [8, 4, 5, 3, 4, 6]

superList : [[Z, A, Y, B], [M, A, N, Y]]

superList Count : 2

superListCombined : [Z, A, Y, B, M, A, N, Y]

superListCombinedLowercase : [z, a, y, b, m, a, n, y]

superListCombinedTwo : [Z, A, Y, B, M, A, N, Y]

dataLengthUsingFlatMapToInt : [1, 1, 1, 1, 1, 1, 1, 1]

sequential -> 0

sequential -> 1

sequential -> 2

sequential -> 3

sequential -> 4

parallel forEach -> 2

parallel forEach -> 3

parallel forEach -> 0

parallel forEach -> 1

parallel forEach -> 4

parallel forEachOrdered -> 0

parallel forEachOrdered -> 1

parallel forEachOrdered -> 2

parallel forEachOrdered -> 3

parallel forEachOrdered -> 4

averageOfStringLength : 5.0

countOfStringLength : 6

joinedString : SubhajitAsimSumanRamAsimSourav

joinedStringDelimeter : Subhajit,Asim,Suman,Ram,Asim,Sourav

joinedStringDelimeterSufPref : prefix : Subhajit,Asim,Suman,Ram,Asim,Sourav : suffix

mapStringLength : {Suman=5, Subhajit=8, Asim=4, Sourav=6, Ram=3}

setString : [Subhajit, Suman, Asim, Sourav, Ram]

stringLengthList : [8, 4, 5, 3, 4, 6]

maxString : Suman

minString : Asim

groupByFunctionCollector : {A=2, R=1, S=3}

groupByFunction : {A=[Asim, Asim], R=[Ram], S=[Subhajit, Suman, Sourav]}

groupByFunctionSupplierCollector : {A=2, R=1, S=3}