Local Variable

Var that is declared inside a methos or block.

Lambda restrictions on local variables:

* Not allowed to use the same local variable name as lambda parameters or inside the lambda body.
* Not allowed to re-assign a value to local variable.

No restrictions on instance variables.

* Introduced in java8.
* Bulk operations on collections
* Parallel operations

Map()

Used to change one type to another(transform)

flatMap():

to transform from List<List<>> to List

distint():

return unique elements

count():

returns a long value with count

sorted():

sorts elements

Also used to sort in customized way by passing Comparator to it.

sorted(Comparator.comparing(Student::getName)

reduce():

terminal operation.

Used to reduce the contents of a stream to a single value.

* First parameter : initial value
* second parameter: BinaryOperator<T>

if you want to perform any MIN or MAX finding on integer list don’t use initial value instead use Optional.

If you use initial value, if the list is empty then that initial value will be returned which is incorrect.

Limit(n):

Limits the n number of elements to be processed in the steam

Skip(n):

Skips the n number of elements from the stream.

All the below functions takes in a Predicate as an input and returns a Boolean as an output.

* anyMatch()fin
* allMatch()
* noneMatch()

anyMatch():

Returns true if any one of the elements matches the predicate, otherwise false.

allMatch():

Returns true if all the elements in the streams matches the predicate, Otherwise false.

noneMatch():

Just opposite to allMatch(). Return true if none of the element in the stream matches the predicate, Otherwise false.

findFirst() and findAny():

* used to find an element in the stream
* both the functions will return the result of type Optional
* findFirst(): 🡪 returns first element in the stream
* findAny(): 🡪 returns the first encountered element in the stream

Create Streams:

* of()

Creates a Stream of certain values passes to this method

* iterate()
* generate()

both used to create infinite streams

Terminal Operations:

Collect():

Takes input type of Collector

1. joining():

Collector performs the string concatenation on the elements in the stream.

3 overloaded methods

1. Counting():

Used to count.

1. Mapping():

This collector applies a transformation function first and then collects the data in a collection (could be any type of collection)

1. minBy() & maxBy():

returns Optional

1. summingInt() & averagingInt():

summingInt ->collectors returns the sum as a result

averagingInt ->collectors returns the average as a result

1. groupingBy():

Collector is equivalent to the groupBy() operation in SQL.

Used to group the elements based on a property

Output is -> Map<K,V>

3 version:

groupingBy(classifier)

groupingBy(classifier,downstream)

groupingBy(classifer,supplier,downstream)

1. partitionBy():

this collector is also a kind of groupingBy().

This will accept a predicate as an input.

Return type 🡪 map<k,v>

The key of the return type is going to be Boolean.

2 versions

partitionBy(Predicate)

partitionBy(Predicate,Downstream)//downstream could be any collector