Data Analysis
(Stream and Parallel Processing 3)

Fall, 2020

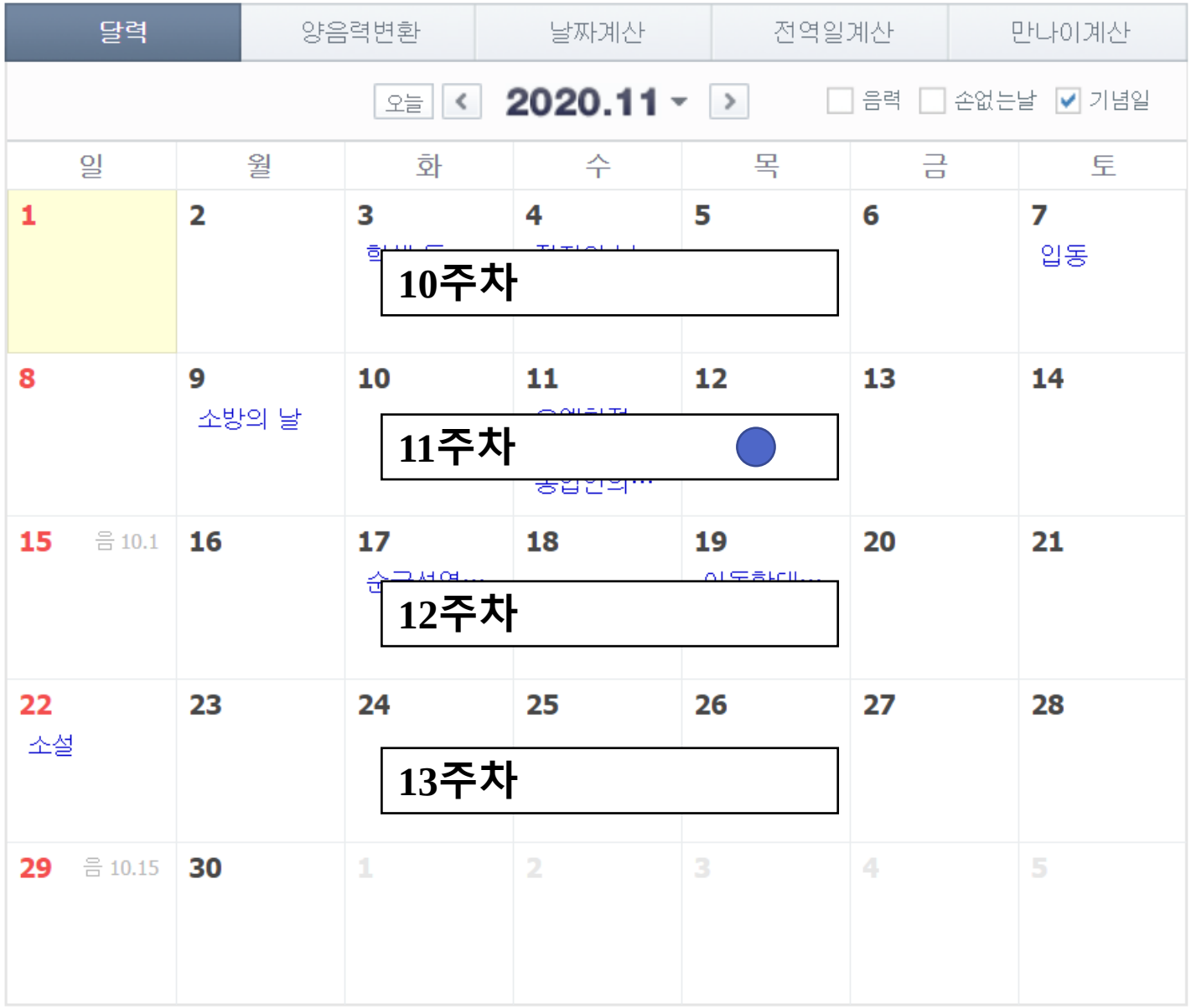
Calendar


달력
양음력변환
날짜계산
전역일계산
만나이계산
오늘
2020.09
음력
손없는날
기념일
일
월
화
수
목
금
토
30
31
1
소개
2
음 7.15
3
환경
세팅
4
지식재산…
5
6
7
백로
8
복습
1
9
10
복습
2
11
12
13
14
15
16
17
음 8.1
18
19
청년의 날
3주차
20
21
치매극복…
22
23
24
25
26
4주차
27
28
29
30
1
2
3
5주차

Calendar


달력
양음력변환
날짜계산
전역일계산
만나이계산
오늘
2020.10
음력
손없는날
기념일
일
월
화
수
목
금
토
27
28
29
30
1 음 8.15
추석
국군의 날
2
노인의 날
3
개천절
4
5
세계 한…
6
7
8
9
한글날
10
6주차
11
12
13
14
15
16
부마민주…
17 음 9.1
문화의 날
7주차
18
19
20
21
22
23
상강
24
국제연합일
8주차: 중간고사
25
독도의날
중양절
26
27
28
29
30
31 음 9.15
9주차

Calendar


달력
양음력변환
날짜계산
전역일계산
만나이계산
오늘
2020.11
음력
손없는날
기념일
일
월
화
수
목
금
토
1
2
3
4
5
6
7
입동
10주차
8
9
소방의 날
10
11
12
13
14
11주차
15
음 10.1
16
17
18
19
20
21
12주차
22
소설
23
24
25
26
27
28
13주차
29
음 10.15
30
1
2
3
4
5

Calendar


달력
양음력변환
날짜계산
전역일계산
만나이계산
오늘
2020.12
음력
손없는날
기념일
일
월
화
수
목
금
토
29
30
1
2
3
4
5
무역의 날
14주차
6
7
대설
8
9
10
11
12
15주차
13
14
15
음 11.1
16
17
18
19
16주차: 기말고사 주간
20
21
동지
22
23
24
25
성탄절
26
27
원자력의…
28
29
음 11.15
30
31
1
2

Table of Contents

Stream API Methods

- Boolean anyMatch(Predicate)
  - Returns whether all elements of this stream match the provided predicate.May not evaluate the predicate on all elements if not necessary fordetermining the result. If the stream is empty then true is returned and the predicate is not evaluated.

- Practice #1 using map and Collectors.toList
  - Make 0 – 99 ArrayList
  - Test if all the elements are larger than 50

Stream API Methods

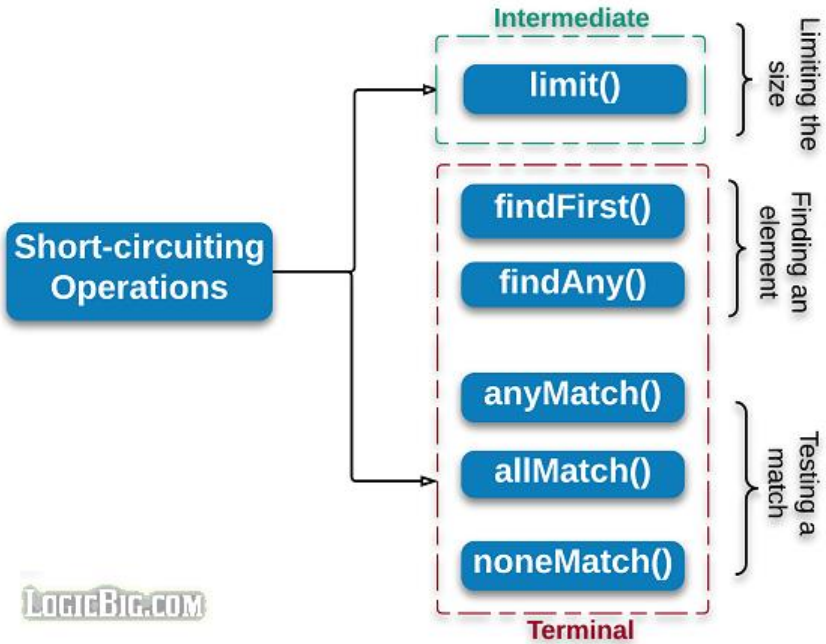
```
public static void main(String[] args) {

int x = 0;
if( x != 0 && 100 / x > 100) {
  System.out.println("[redacted]");
}

  System.out.println("[redacted]");
}
```

- [redacted]
  - means that when evaluating boolean expressions (logical AND and OR) you can stop as soon as you find the first condition which satisfies or negates the expression.

Stream API Methods



https://www.logicbig.com/tutorials/core-java-tutorial/java-util-stream/short-circuiting.html

- **anyMatch**: Returns whether any elements of this stream match the providedpredicate. May not evaluate the predicate on all elements if notnecessary for determining the result. If the stream is empty then false is returned and the predicate is not evaluated.
- **noneMatch**: Returns whether no elements of this stream match the provided predicate.May not evaluate the predicate on all elements if not necessary fordetermining the result. If the stream is empty then true isreturned and the predicate is not evaluated.

Stream API Methods

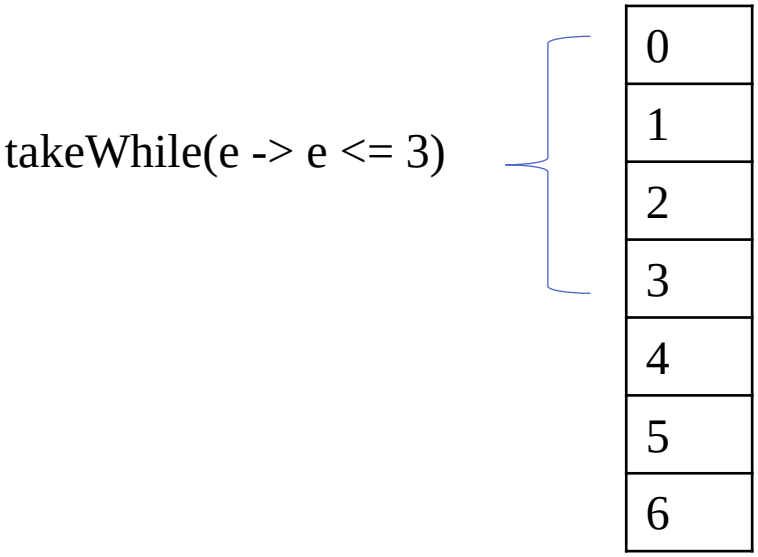
- Stream<Integer> peek(Consumer)
  - Returns a stream consisting of the elements of this stream, additionallyperforming the provided action on each element as elements are consumedfrom the resulting stream.

- Unlike forEach, it is intermediate operator

- Practice #2 using peek and map and collect
  - Make 0 – 99 ArrayList
  - Print out each element
  - Map each element to X3
  - Collect them to list

Stream API Methods

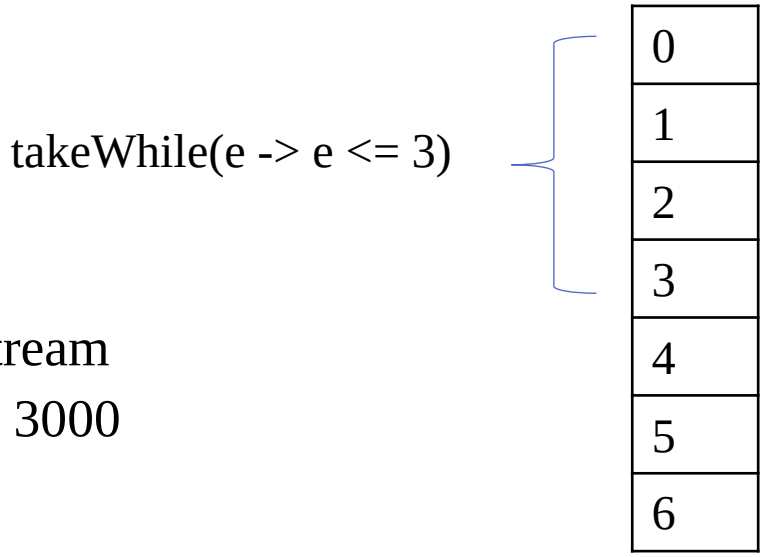
- Stream takeWhile(Predicate)
  - Returns a stream consisting of the elements of this stream, additionally performing the provided action on each element as elements are consumed from the resulting stream.



- Practice #3 using takeWhile and collect
  - Make 0 – 99 ArrayList
  - Take elements until that is less than 10
  - Collect them to list

Stream API Methods

- Stream takeWhile(Predicate)
  - Returns a stream consisting of the elements of this stream, additionally performing the provided action on each element as elements are consumed from the resulting stream.



- Practice #4 using takeWhile and collect
  - Make 2000 – 9999 HashSet Parallel Stream
  - Take elements until that is greater than 3000
  - Collect them to list
  - Print out the size of the list

Stream API Methods

- Stream takeWhile(Predicate)
  - Returns a stream consisting of the elements of this stream, additionally performing the provided action on each element as elements are consumed from the resulting stream.

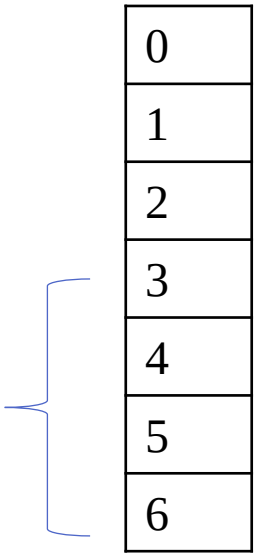
- Practice #5 using takeWhile and collect
  - Make 0 – 99 ArrayList
  - Take elements until that is greater than 3?
  - Collect them to list

Stream API Methods

- Stream dropWhile(Predicate)
  - Returns, if this stream is ordered, a stream consisting of the remainingelements of this stream after dropping the longest prefix of elementsthat match the given predicate. Otherwise returns, if this stream isunordered, a stream consisting of the remaining elements of this streamafter dropping a subset of elements that match the given predicate.

- Practice #6 using dropWhile and collect
  - Make 0 – 99 ArrayList
  - Take elements until that is less than 70
  - Collect them to list

takeWhile(e -> e < 3)



Stream API Methods

- Stream dropWhile(Predicate)
  - Returns, if this stream is ordered, a stream consisting of the remainingelements of this stream after dropping the longest prefix of elementsthat match the given predicate. Otherwise returns, if this stream isunordered, a stream consisting of the remaining elements of this streamafter dropping a subset of elements that match the given predicate.

- Practice #7 using dropWhile and collect
  - Make 0 – 99 ArrayList
  - Take elements until that is greater than 30
  - Collect them to list

Stream API Methods

- Stream sort(comparator)
  - Returns a stream consisting of the elements of this stream, sortedaccording to the provided Comparator.

- Practice #8-1 (revisit sort()) sort
  - Make ArrayList 0 to 999 random 100 numbers
  - sort
  - Collect them to list

Stream API Methods

- Stream sort(comparator)
  - Returns a stream consisting of the elements of this stream, sortedaccording to the provided Comparator.

- Practice #8-2 (revisit sort()) sort
  - Make ArrayList 0 to 999 random left and right 100 emails
  - sort
  - Collect them to list

Stream API Methods

- Stream sort(comparator)
  - Returns a stream consisting of the elements of this stream, sortedaccording to the provided Comparator.

- Practice #8-3 sort (comparator)
  - Make ArrayList 0 to 999 random left and right 100 emails
  - Sort based on left value
  - Collect them to list

Stream API Methods

- void close()
  - Closes this stream, causing all close handlers for this stream pipeline to be called.
- Stream onClose(Runnable)
  - Returns an equivalent stream with an additional close handler. Closehandlers are run when the close() method is called on the stream, and are executed in the order they were added.

- Practice #9 onClose and close
  - Make ArrayList of 0 to 5
  - Print out each element
  - Collect each element to list
  - Print out list
  - Then close → print out x

Stream API Methods

- Stream parallel()
  - Returns an equivalent stream that is parallel. May returnitself, either because the stream was already parallel, or becausethe underlying stream state was modified to be parallel.
- Stream sequential()
  - Returns an equivalent stream that is sequential. May returnitself, either because the stream was already sequential, or becausethe underlying stream state was modified to be sequential.
- Boolean isParallel()
  - Returns whether this stream, if a terminal operation were to be executed,would execute in parallel. Calling this method after invoking anterminal stream operation method may yield unpredictable results.

- Practice #10
  - Make stream
  - Invoke parallel()
  - Check isParallel
  - Invoke sequential()
  - Check sequential

Stream API Methods

- IntStream mapToInt(ToIntFunction)
  - Returns an IntStream consisting of the results of applying thegiven function to the elements of this stream.

- DoubleStream mapToDouble(ToDoubleFunction)
  - Returns a DoubleStream consisting of the results of applying thegiven function to the elements of this stream.

- LongStream mapToLong(ToLongFunction)
  - Returns a LongStream consisting of the results of applying thegiven function to the elements of this stream.

- These Stream API maps object to Integer/Double/Long that Stream recognizes

- Why?

Stream API Methods

- IntStream mapToInt(ToIntFunction)
- DoubleStream mapToDouble(ToDoubleFunction)
- LongStream mapToLong(ToLongFunction)

- Why? Convenience
  - IntStream.sum()
  - IntStream.average()
  - IntStream.max()
  - IntStream.min()
  - IntStream.summaryStatistics()

- Practice #11
  - Make stream of 0 to 99 Integer
  - mapToInt
  - Do average or summaryStatistics()

Stream API Methods

- Stream<Integer> IntStream.boxed()

- Unboxing
  - Stream → IntStream
- Boxing
  - IntStream → Stream<Integer>

- DoubleStream IntStream.mapToDouble(IntToDoubleFunction)
- LongStream IntStream.mapToLong(IntToLongFunction)
- Stream<U> IntStream.mapToObject(IntFunction)

Stream API Methods

- IntStreamflatMapToInt(Function<U, ? extends IntStream> mapper)
  - Returns an IntStream consisting of the results of replacing eachelement of this stream with the contents of a mapped stream produced byapplying the provided mapping function to each element. Each mappedstream is closed after itscontents have been placed into this stream. (If a mapped stream is null an empty stream is used, instead.)
- DoubleStream flatMapToDouble(Function<U, ? extends DoubleStream>)
  - Returns an DoubleStream consisting of the results of replacingeach element of this stream with the contents of a mapped stream producedby applying the provided mapping function to each element. Each mappedstream is closed after itscontents have placed been into this stream. (If a mapped stream is null an empty stream is used, instead.)
- LongStream flatMapToLong(Function<U, ? extends LongStream> mapper)
  - Returns an LongStream consisting of the results of replacing eachelement of this stream with the contents of a mapped stream produced byapplying the provided mapping function to each element. Each mappedstream is closed after itscontents have been placed into this stream. (If a mapped stream is null an empty stream is used, instead.)

Stream API Methods

- IntStreamflatMapToInt(Function<U, ? extends IntStream> mapper)
- DoubleStream flatMapToDouble(Function<U, ? extends DoubleStream>)
- LongStream flatMapToLong(Function<U, ? extends LongStream> mapper)

- Practice #12
  - Make HashMap<String, ArrayList<Integer>> of students' grade
    - One student has 5 points
  - Get entrySet and stream
  - Flatten the grades via flatMapToInteger
  - Get the average

Stream API Methods

- <R> R collect(Supplier<R> supplier, BiConsumer<R, ? Super T> accumulator, BiConsumer<R,R> combiner>
- R: type of the result
- **Supplier**: a function that creates a new result container (mutable object) . For a parallel execution, this function may be called multiple times. It must return a fresh value each time.
- **Accumulator**: a function for incorporating an additional element into a result.
- **Combiner**: a function for combining two values, used in parallel stream, combines the results received from different threads.

Stream API Methods

- <R> R collect(Supplier<R> supplier, BiConsumer<R, ? Super T> accumulator, BiConsumer<R,R> combiner>
- R: type of the result
- **Supplier**: a function that creates a new result container (mutable object) . For a parallel execution, this function may be called multiple times. It must return a fresh value each time.
- **Accumulator**: a function for incorporating an additional element into a result.
- **Combiner**: a function for combining two values, used in parallel stream, combines the results received from different threads.

Example: Concatenate Integers

1	2	3	4	5	6	7

“1234567”

Stream API Methods

- <R> R collect(Supplier<R> supplier, BiConsumer<R, ? Super T> accumulator, BiConsumer<R,R> combiner>
- R: type of the result
- **Supplier**: a function that creates a new result container (mutable object) . For a parallel execution, this function may be called multiple times. It must return a fresh value each time.
- **Accumulator**: a function for incorporating an additional element into a result.
- **Combiner**: a function for combining two values, used in parallel stream, combines the results received from different threads.

Example: Concatenate Integers

1	2	3	4	5	6	7

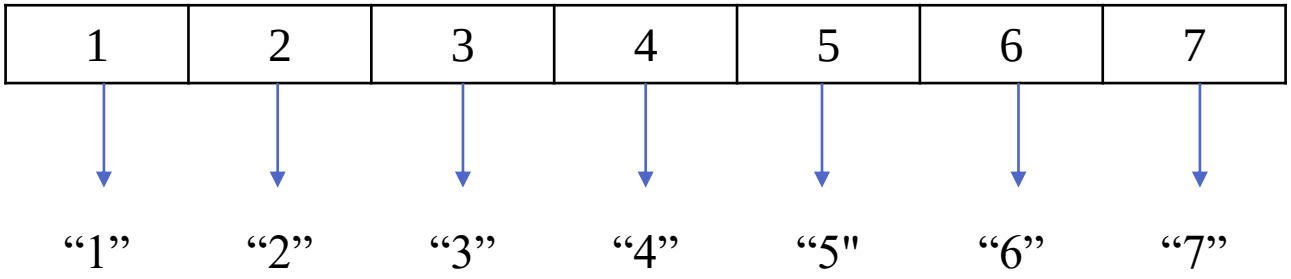
Step 1: Make a null string to concatenate each integer (supplier)

“”

Stream API Methods

- <R> R collect(Supplier<R> supplier, BiConsumer<R, ? Super T> accumulator, BiConsumer<R,R> combiner>
- R: type of the result
- **Supplier**: a function that creates a new result container (mutable object) . For a parallel execution, this function may be called multiple times. It must return a fresh value each time.
- **Accumulator**: a function for incorporating an additional element into a result.
- **Combiner**: a function for combining two values, used in parallel stream, combines the results received from different threads.

Example: Concatenate Integers

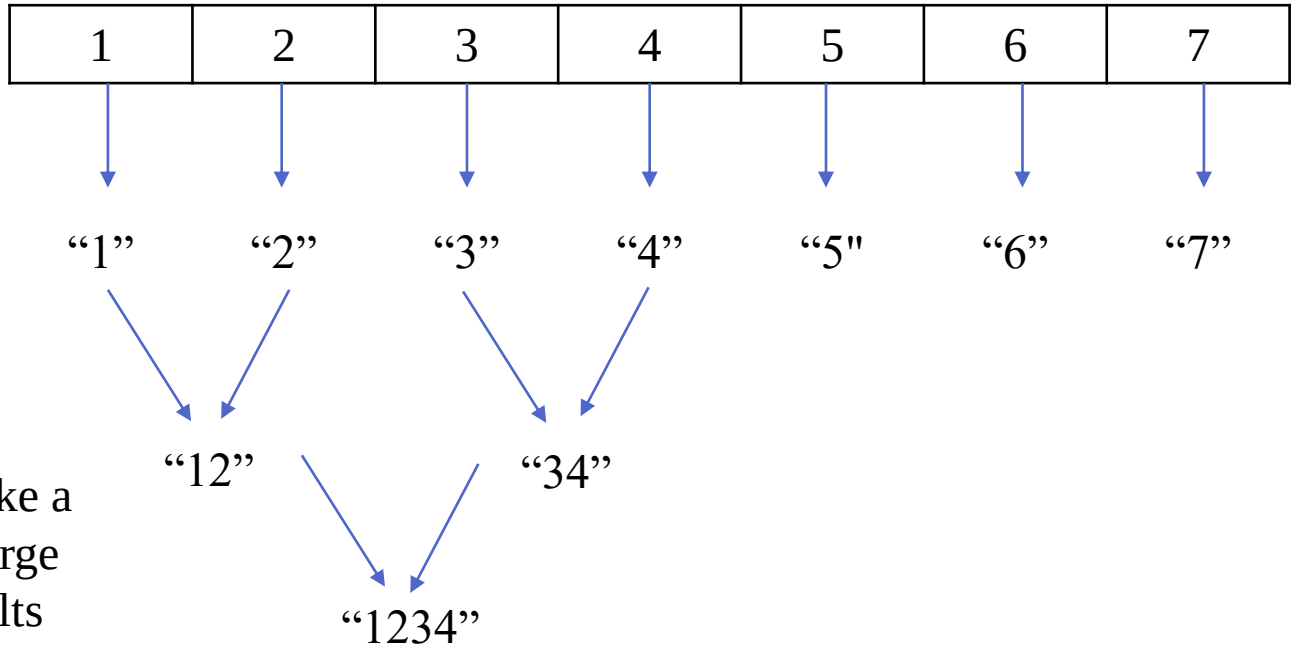


Step 2: Make a logic to add each integer to the string (accumulator)

Stream API Methods

- <R> R collect(Supplier<R> supplier, BiConsumer<R, ? Super T> accumulator, BiConsumer<R,R> combiner>
- R: type of the result
- **Supplier**: a function that creates a new result container (mutable object) . For a parallel execution, this function may be called multiple times. It must return a fresh value each time.
- **Accumulator**: a function for incorporating an additional element into a result.
- **Combiner**: a function for combining two values, used in parallel stream, combines the results received from different threads.

Example: Concatenate Integers



Step 3: Make a logic to merge partial results

Stream API Methods

- <R> R collect(Supplier<R> supplier, BiConsumer<R, ? Super T> accumulator,
  BiConsumer<R,R> combiner>
- R: type of the result
- **Supplier**: a function that creates a new result container (mutable object) . For a parallel execution, this function may be called multiple times. It must return a fresh value each time.
- **Accumulator**: a function for incorporating an additional element into a result.
- **Combiner**: a function for combining two values, used in parallel stream, combines the results received from different threads.

- Practice #13
  - Make stream of 0 to 99
  - Collect with int to string concatenation

Do it with StringBuilder

Stream API Methods

- <R> R collect(Supplier<R> supplier, BiConsumer<R, ? Super T> accumulator, BiConsumer<R,R> combiner>
- R: type of the result
- **Supplier**: a function that creates a new result container (mutable object) . For a parallel execution, this function may be called multiple times. It must return a fresh value each time.
- **Accumulator**: a function for incorporating an additional element into a result.
- **Combiner**: a function for combining two values, used in parallel stream, combines the results received from different threads.

Example: Sum Integers

1	2	3	4	5	6	7

28

Stream API Methods

- <R> R collect(Supplier<R> supplier, BiConsumer<R, ? Super T> accumulator, BiConsumer<R,R> combiner>
- R: type of the result
- **Supplier**: a function that creates a new result container (mutable object) . For a parallel execution, this function may be called multiple times. It must return a fresh value each time.
- **Accumulator**: a function for incorporating an additional element into a result.
- **Combiner**: a function for combining two values, used in parallel stream, combines the results received from different threads.

Example: Sum Integer

1	2	3	4	5	6	7

Step 1: Make a base integer instance to sum each integer (supplier)

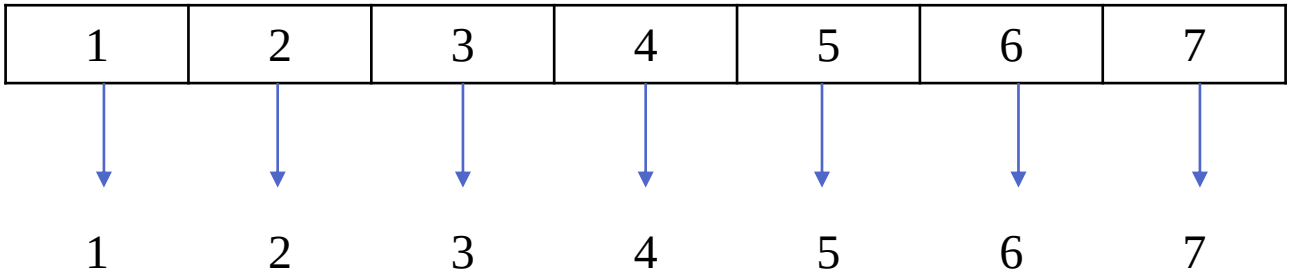
“”

You can use AtomicInteger

Stream API Methods

- <R> R collect(Supplier<R> supplier, BiConsumer<R, ? Super T> accumulator, BiConsumer<R,R> combiner>
- R: type of the result
- **Supplier**: a function that creates a new result container (mutable object) . For a parallel execution, this function may be called multiple times. It must return a fresh value each time.
- **Accumulator**: a function for incorporating an additional element into a result.
- **Combiner**: a function for combining two values, used in parallel stream, combines the results received from different threads.

Example: Concatenate Integers

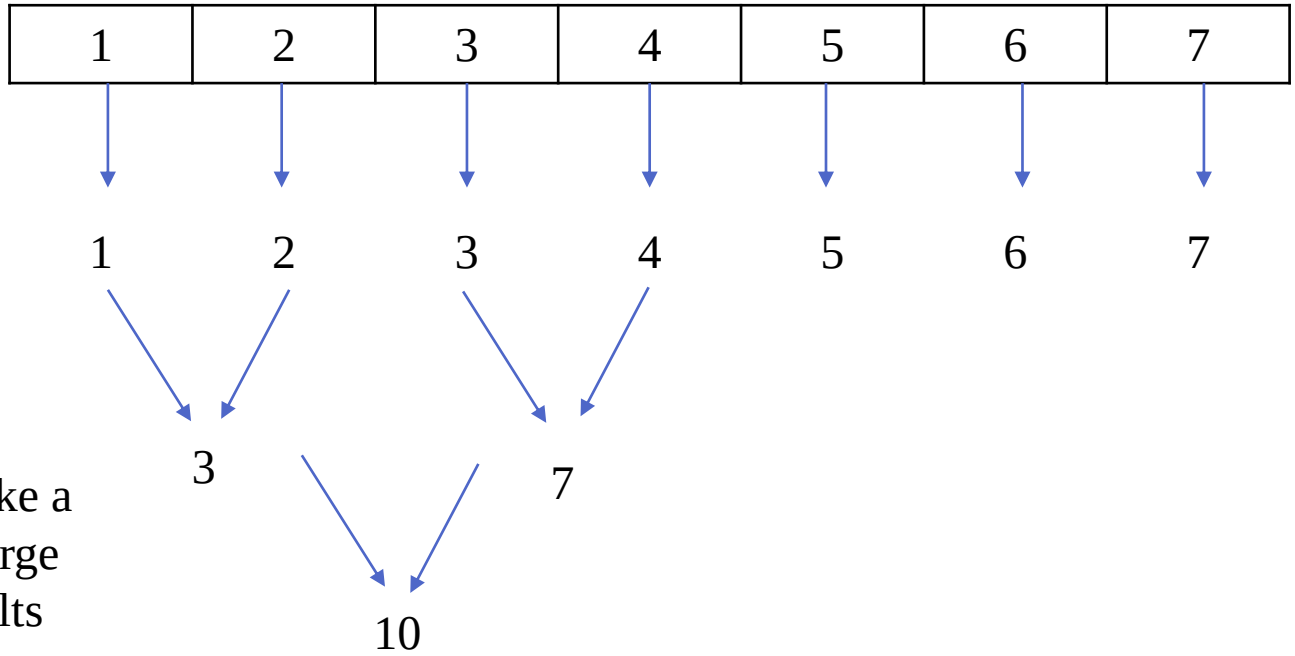


Step 2: Make a logic to add each integer to the atomic integer (accumulator)

Stream API Methods

- <R> R collect(Supplier<R> supplier, BiConsumer<R, ? Super T> accumulator, BiConsumer<R,R> combiner>
- R: type of the result
- **Supplier**: a function that creates a new result container (mutable object) . For a parallel execution, this function may be called multiple times. It must return a fresh value each time.
- **Accumulator**: a function for incorporating an additional element into a result.
- **Combiner**: a function for combining two values, used in parallel stream, combines the results received from different threads.

Example: Concatenate Integers



Step 3: Make a logic to merge partial results

Stream API Methods

- <R> R collect(Supplier<R> supplier, BiConsumer<R, ? Super T> accumulator,
  BiConsumer<R,R> combiner>
- R: type of the result
- **Supplier**: a function that creates a new result container (mutable object) . For a parallel execution, this function may be called multiple times. It must return a fresh value each time.
- **Accumulator**: a function for incorporating an additional element into a result.
- **Combiner**: a function for combining two values, used in parallel stream, combines the results received from different threads.

- Practice #14
  - Make stream of 0 to 99
  - Collect with your own collector

Do it with AtomicInteger

Stream API Methods

- Practice #15 with email data
  - Make ArrayList<Integer> having left IDs and right IDs with redundancy
  - Compute occurrence for each IDs with the following steps:

Stream<String>

# …	# …	# …	# …	# …	0 1	0 2	0 4	1 2	3 4

Filter starts with #

Stream<String>

0 1	0 2	0 4	1 2	3 4

flatten

Stream<String>

0	1	0	2	0	4	1	2	3	4

map

Stream<Integer>

0	1	0	2	0	4	1	2	3	4

collect

HashMap<Integer, Integer>

<0,3>	<1,2>	<2,2>	<3,1>	<4,2>

Stream API Methods

- Collector with built-in collectors
  - Easy - Decimal
    - Collect(Collectors.averagingDouble(ToDoubleFunction))
    - Collect(Collectors.averagingInt(ToIntFunction))
    - Collect(Collectors.averagingLong(ToLongFunction))
    - Collect(Collectors.summingDouble(ToDoubleFunction))
    - Collect(Collectors.summingInt(ToIntFunction))
    - Collect(Collectors.summingInt(ToLongFunction))
    - Collect(Collectors.summarizingDouble(ToDoubleFunction))
    - Collect(Collectors.summarizingInt(ToIntFunction))
    - Collect(Collectors.summarizingInt(ToLongFunction))

    - Collect(Collectors.maxBy(Comparator))
    - Collect(Collectors.minBy(Comparator))

- Practice #16
  - Make ArrayList<Integer> from 0 to 99
  - Apply the above methods

Stream API Methods

- Collector with built-in collectors
  - Easy – String
    - Collectors.joining();
    - Collectors.joining(Delimiter);
    - Collectors.joining(Delimiter, Prefix, Suffix);

- Practice #17
  - Make ArrayList<Integer> from 0 to 4
  - Make “01234”
  - Make “0,1,2,3,4”
  - Make “$0,1,2,3,4^”

Stream API Methods

- Collector with built-in collectors
  - Advanced
    - Map<K, List<K>> collect(Collectors.groupingBy(Function<K, K> classifier))
      - Returns a Collector implementing a "group by" operation on input elements of type T, grouping elements according to a classification function, and returning the results in a Map.

0	1	2	1	2	3	2	3	4

classifier: how to group?

<0, [0]>	<1, [1,1]>	<2, [2,2,2]>	<3, [3,3]>	<4, [4]>

Stream API Methods

- Collector with built-in collectors
  - Advanced
    - Map<K, List<K>> collect(Collectors.groupingBy(Function<K, K> classifier))
      - Returns a Collector implementing a "group by" operation on input elements of type T, grouping elements according to a classification function, and returning the results in a Map.

- Practice #18
  - Make ArrayList<Integer> of 100 random numbers from 0 to 4
  - groupBy each number Map<Integer, List<Integer>>

Stream API Methods

- Collector with built-in collectors
  - Advanced
    - Map<K, List<K>> collect(Collectors.groupingBy(Function<K, K> classifier))
      - Returns a Collector implementing a "group by" operation on input elements of type T, grouping elements according to a classification function, and returning the results in a Map.

- Practice #19 with email data
  - Collect ArrayList<Email>
  - Group by from ID
  - Map<Integer, List<Email>>

Stream API Methods

- Collector with built-in collectors
  - Advanced
    - Map<K, List<K>> collect(Collectors.groupingBy(Function<K, K> classifier))
      - Returns a Collector implementing a "group by" operation on input elements of type T, grouping elements according to a classification function, and returning the results in a Map.

- Practice #20
  - Make ArrayList<Integer> of 100 random numbers from 0 to 99
  - Group by %5
  - Map<Integer, List<Integer>>

Can we handle this?

Stream API Methods

- Collector with built-in collectors
  - Advanced
    - Map<K, D> collect(Collectors.groupingBy(Function<K, K> classifier, Collector<T,A,D> downstream))
      - Returns a Collector implementing a cascaded "group by" operationon input elements of type T, grouping elements according to aclassification function, and then performing a reduction operation onthe values associated with a given key using the specified downstream Collector.
    - Using Collector Collectors.mapping(Function mapper, Collector downstream)

0	1	2	1	2	3	2	3	4

classifier: how to group?

downstream: List to D

Collectors.mapping(, How to produce)

Each elem

<0, [0]>	<1, [1,1]>	<2, [2,2,2]>	<3, [3,3]>	<4, [4]>

Stream API Methods

- Collector with built-in collectors
  - Advanced
    - Map<K, D> collect(Collectors.groupingBy(Function<K, K> classifier, Collector<T,A,D> downstream))
      - Returns a Collector implementing a cascaded "group by" operationon input elements of type T, grouping elements according to aclassification function, and then performing a reduction operation onthe values associated with a given key using the specified downstream Collector.
    - Using Collector Collectors.mapping(Function mapper, Collector downstream)

- Practice #21
  - Make ArrayList<Integer> of 100 random numbers from 0 to 99
  - Group by %5 and its count, summarizingDouble, etc.
  - Map<Integer, Integer>

Stream API Methods

- Collector with built-in collectors
  - Advanced
    - Map<K, D> collect(Collectors.groupingBy(Function<K, K> classifier, Collector<T,A,D> downstream))
      - Returns a Collector implementing a cascaded "group by" operationon input elements of type T, grouping elements according to aclassification function, and then performing a reduction operation onthe values associated with a given key using the specified downstream Collector.
    - Using Collector Collectors.mapping(Function mapper, Collector downstream)

- Practice #21
  - Make ArrayList<Integer> of 100 random numbers from 0 to 99
  - Group by %5 and its count, summarizingDouble, etc.
  - Map<Integer, Integer>

Stream API Methods

- Collector with built-in collectors
  - Advanced
    - Map<K, D> collect(Collectors.groupingBy(Function<K, K> classifier, Collector<T,A,D> downstream))
      - Returns a Collector implementing a cascaded "group by" operationon input elements of type T, grouping elements according to aclassification function, and then performing a reduction operation onthe values associated with a given key using the specified downstream Collector.
    - Using Collector Collectors.filtering(Predicate predicate, Collector downstream)
      - Adapts a Collector to one accepting elements of the same type T by applying the predicate to each input element and onlyaccumulating if the predicate returns true.

- Practice #22
  - Make ArrayList<Integer> of 100 random numbers from 0 to 99
  - Group by %4 and filter even numbers, reducing its count
  - Map<Integer, Integer>

Stream API Methods

- Collector with built-in collectors
  - Advanced
    - Map<K, D> collect(Collectors.groupingBy(Function<K, K> classifier, mapFactory, Collector<T,A,D> downstream))
      - Returns a Collector implementing a cascaded "group by" operationon input elements of type T, grouping elements according to aclassification function, and then performing a reduction operation onthe values associated with a given key using the specified downstream Collector. The Map produced by the Collector is createdwith the supplied factory function.

- Practice #24
  - Make ArrayList<Integer> of 100 random numbers from 0 to 99
  - Group by %4 to TreeMap instead of Map
  - TreeMap<Integer, List<Integer>>

Stream API Methods

- Collector with built-in collectors
  - Advanced
    - Map<K, D> collect(Collectors.groupingBy(Function<K, K> classifier)

- Practice #25 – motivating example
  - Make ArrayList<Integer> of 100 random numbers from 0 to 99
  - Group by %2 and reducing count
  - Map<Integer, Integer>

Stream API Methods

- Collector with built-in collectors
  - Advanced
    - Map<Boolean, List<K>> collect(Collectors.partitioningBy(Predicate predicate)
      - Returns a Collector which partitions the input elements accordingto a Predicate, reduces the values in each partition according toanother Collector, and organizes them into a Map<Boolean, D> whose values are the result of the downstreamreduction.
    - Map<Boolean, D> collect(Collectors.partitioningBy(Predicate predicate, downstream)

- Practice #26
  - Make ArrayList<Integer> of 100 random numbers from 0 to 99
  - Group by even numbers or odd numbers and reducing count using partitioningBy
  - Map<Boolean, Integer>

Wrap-up

- Stream and Parallel Processing 3 - Advanced