Stream Method

학습내용

1. Stream Method의 소개
2. Stream Method의 활용

학습목표

- Stream에서 지원하는 각각의 method의 개념을 설명할 수 있다.
- Stream에서 지원하는 method를 활용을 할 수 있다.

Intermediate stream method

Stream method	Description
Stream<T> filter(Predicate<? super T> predicate)	Stream의 각 instance를 test하여 성공하면 유지
<R> Stream<R> map(Function<? super T, ? extends R> mapper);	Stream의 각 instance를 mapper를 적용한 결과로 변환
IntStream mapToInt(ToIntFunction<? super T> mapper);	Stream의 각 instance를 정수형의 추가적 연산이 가능한 IntStream으로 변환
<R> Stream<R> flatMap(Function<? super T, ? extends Stream<? extends R>> mapper);	Stream의 각 instance의 mapper를 적용한 Stream들을 Flattening하여 변환
Stream<T> distinct();	Stream의 중복을 제거
Stream<T> sorted();	Stream을 정렬
Stream<T> sorted(Comparator<? super T> comparator);	Stream을 비교할 수 있는 방법 comparator를 통하여 정렬
Stream<T> peek(Consumer<? super T> action);	Stream의 각 instance에 action을 수행하지만 Stream의 구성이 바뀌지 않음
Stream<T> limit(long maxSize);	Stream의 초반 n개의 요소만을 유지
Stream<T> skip(long n)	Stream에서 초반 n개의 요소외의 instance들을 유지
Stream<T> takeWhile(Predicate<? super T> predicate)	Stream에서 predicate을 위배하기 전 까지만 유지
Stream<T> dropWhile(Predicate<? super T> predicate)	Stream에서 predicate을 위배하기 전 까지는 무시

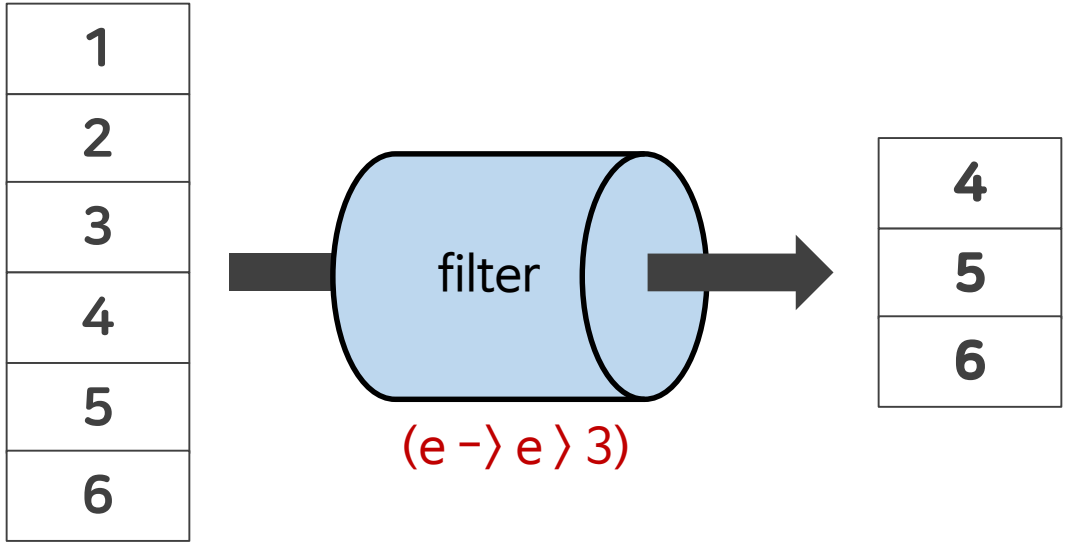
Intermediate stream method

Stream method	Description
long count();	Stream의 instance 개수 반환
Optional<T> min(Comparator<? super T> comparator);	Stream에서 comparator에 의한 최솟값을 찾을 수 있도록 반환
Optional<T> max(Comparator<? super T> comparator);	Stream에서 comparator에 의한 최댓값을 찾을 수 있도록 반환
Optional<T> findFirst();	Stream에서 가장 처음 instance를 가질 수 있도록 반환
Optional<T> findAny();	Stream에서 아무 instance를 가질 수 있도록 반환
boolean anyMatch(Predicate<? super T> predicate);	Stream 전체에 대해 test를 통과한 instance가 하나라도 있으면 true
boolean allMatch(Predicate<? super T> predicate);	Stream 전체에 대해 instance들이 test를 모두 통과해야 true
boolean noneMatch(Predicate<? super T> predicate);	Stream 전체에 대해 instance들이 test를 모두 통과 못해야 true
Object[] toArray();	Stream을 Object[] 배열로 반환
<A> A[] toArray(IntFunction<A[]> generator);	Stream을 A[] 배열로 반환
void forEach(Consumer<? super T> action);	Stream의 각각의 instance에 대해 action 수행
T reduce(T identity, BinaryOperator<T> accumulator);	Stream<T> 전체를 T identity라는 단일 결과값에 Reduction 함
Optional<T> reduce(BinaryOperator<T> accumulator);	Stream<T> 전체를 T 타입의 단일 결과값 Reduction을 가질 수 있도록 반환
<R> R collect(Supplier<R> supplier, BiConsumer<R, ? super T> accumulator, BiConsumer<R, R> combiner);	Stream<T> 전체에 대한 결과를 담을 용기 supplier를 제공하고, instance를 supplier에 담는 방법 (accumulator)과 병렬 처리시 중간 결과 값들을 병합할 수 있는 combiner를 제공하여 모음

filter

Stream method	Description
Stream<T> filter(Predicate <? super T> predicate)	Stream의 각 instance를 test하여 성공하면 유지

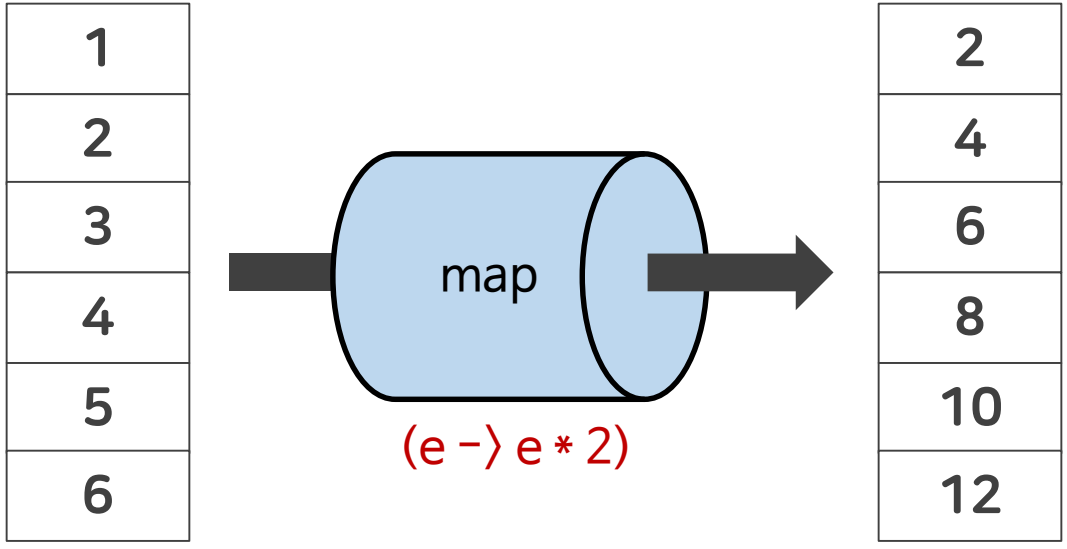
filter



map

Stream method	Description
⟨R⟩ Stream⟨R⟩ map(Function ⟨? super T, ? extends R⟩ mapper);	Stream의 각 instance를 mapper를 적용한 결과로 변환

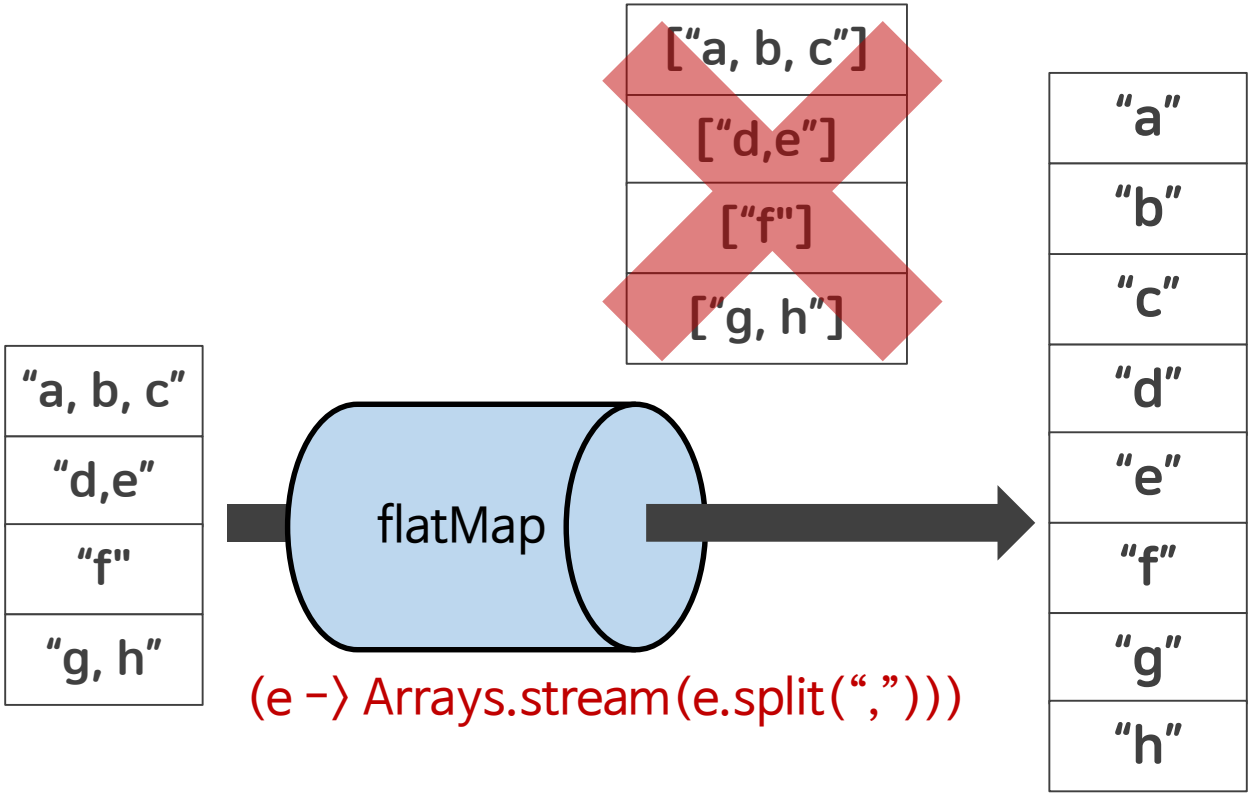
map



Flatmap

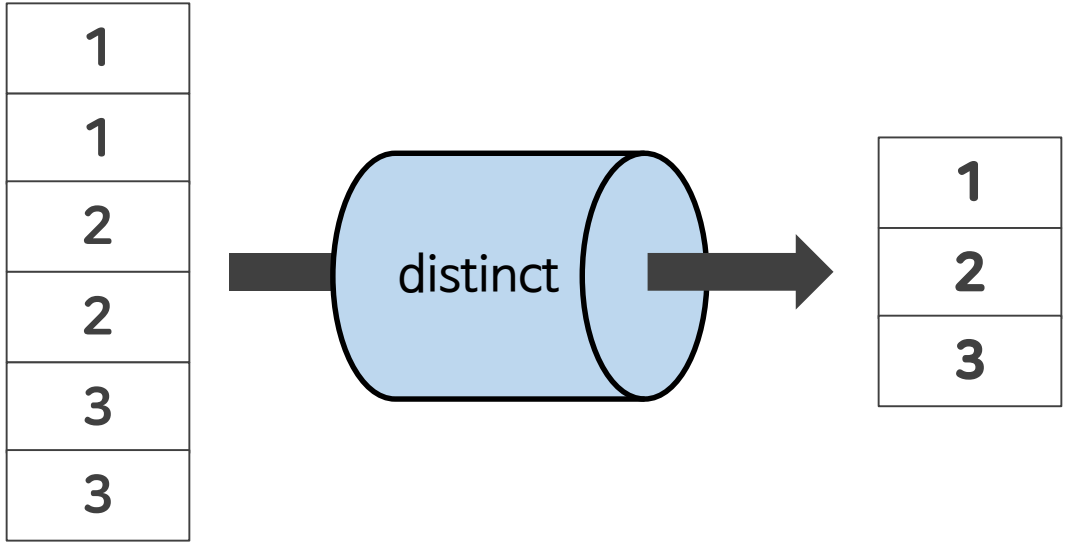
Stream method	Description
⟨R⟩ Stream⟨R⟩ flatMap (Function ⟨? super T, ? extends Stream ⟨? extends R⟩⟩ mapper);	Stream의 각 instance의 mapper를 적용한 Stream들을 Flattening하여 변환

Flatmap


["a, b, c"]
["d,e"]
["f"]
["g, h"]
"a, b, c"
"d,e"
"f"
"g, h"
flatMap
(e -> Arrays.stream(e.split(",")))
"a"
"b"
"c"
"d"
"e"
"f"
"g"
"h"

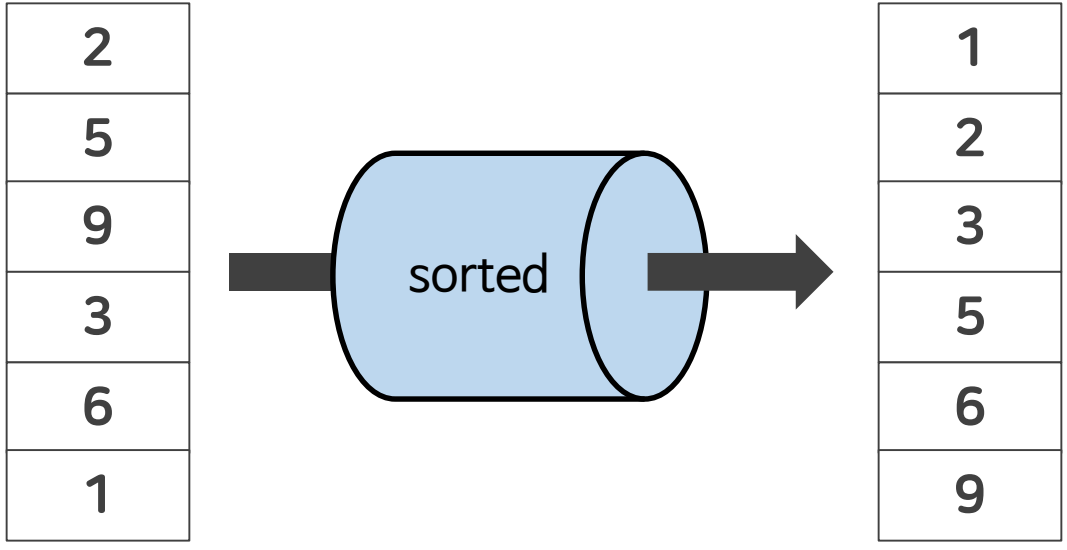
distinct

Stream method	Description
Stream<T> distinct();	Stream의 중복을 제거



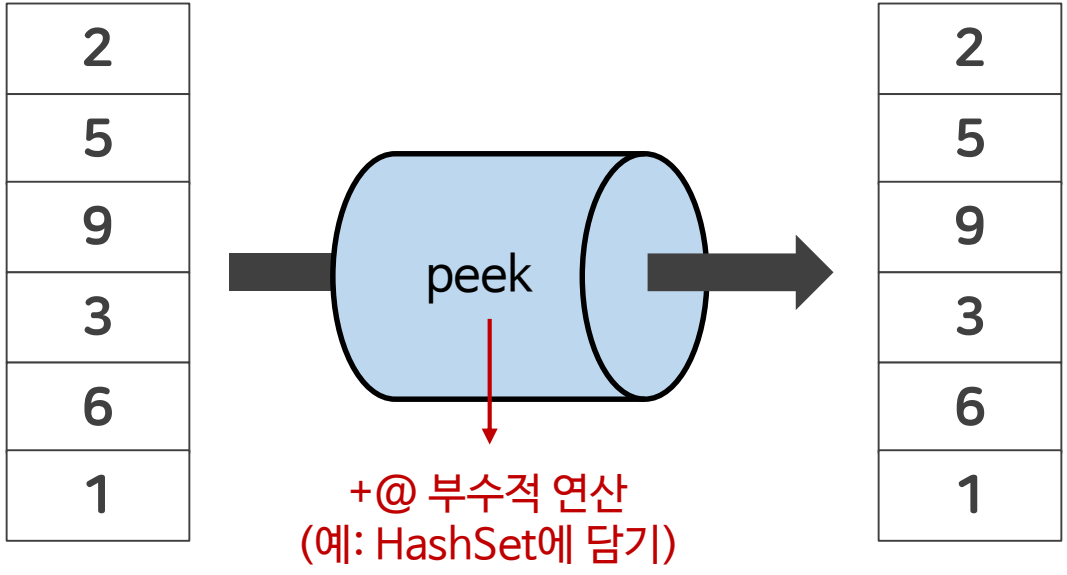
sorted

Stream method	Description
Stream<T> sorted();	Stream을 정렬
Stream<T> sorted(Comparator<? super T> comparator);	Stream을 비교할 수 있는 방법 comparator를 통하여 정렬



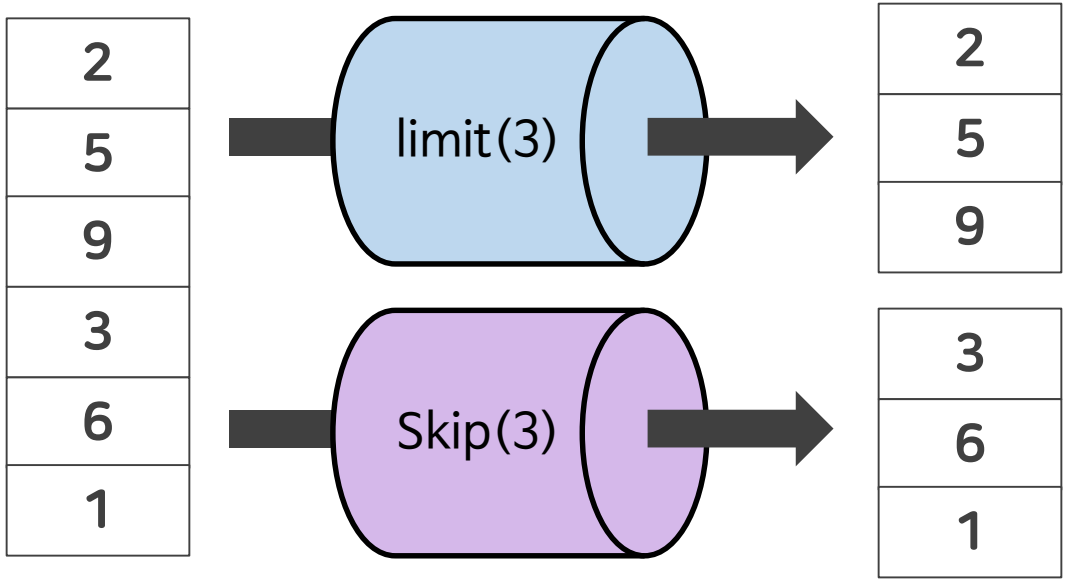
peek

Stream method	Description
Stream<T> peek(Consumer<? super T> action);	Stream의 각 instance에 action을 수행하지만 Stream의 구성이 바뀌지 않음



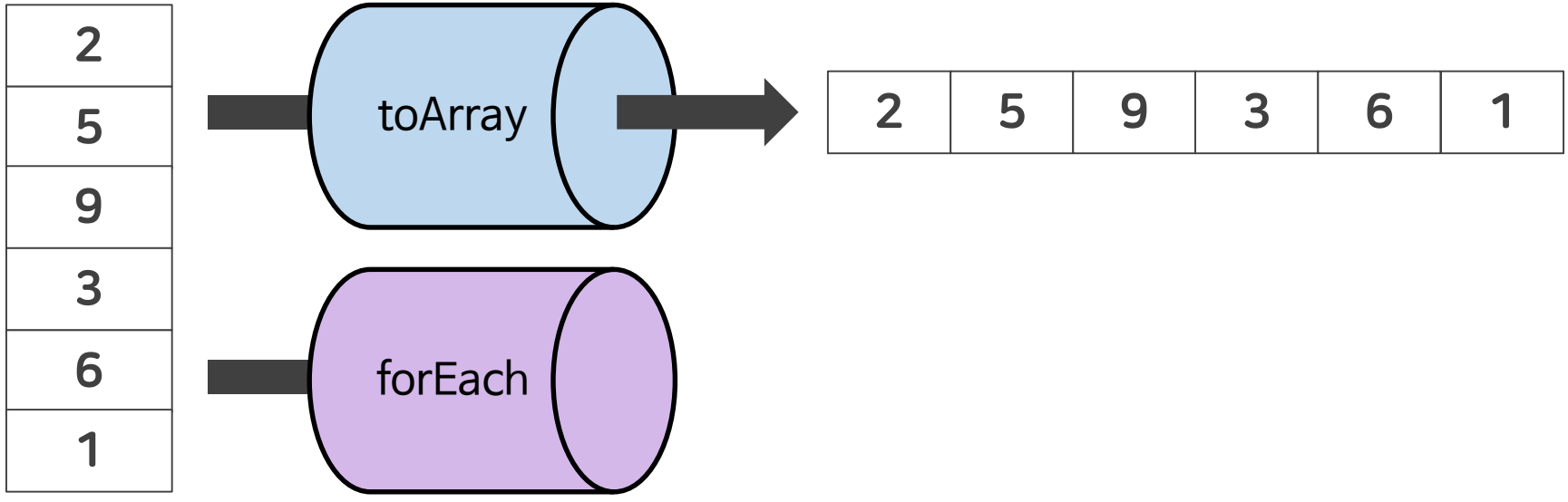
limit, skip

Stream method	Description
Stream<T> limit (long maxSize);	Stream의 초반 n개의 요소만을 유지
Stream<T> skip(long n)	Stream에서 초반 n개의 요소외의 instance들을 유지



toArray, forEach

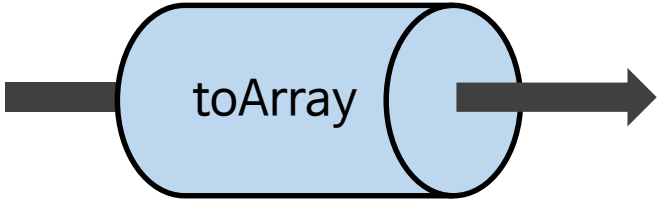
Stream method	Description
Object[] toArray();	Stream을 Object[] 배열로 반환
<A> A[] toArray(IntFunction<A[]> generator);	Stream을 A[] 배열로 반환
void forEach(Consumer<? super T> action);	Stream의 각각의 instance에 대해 action 수행



reduce

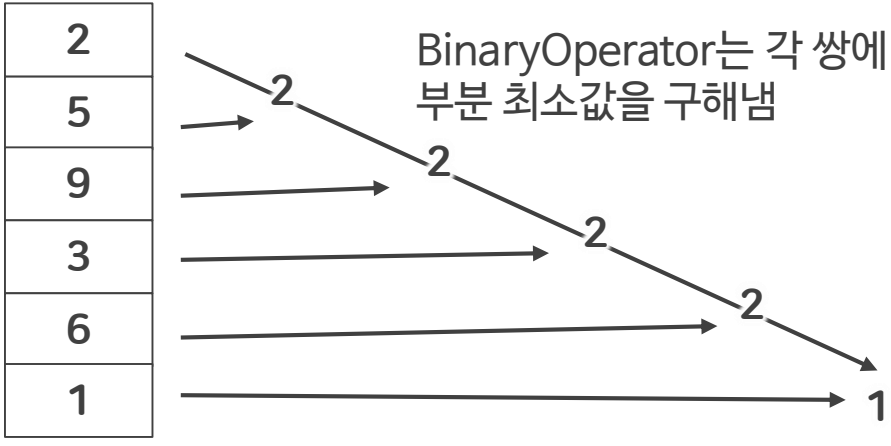
Stream method	Description
T reduce(T identity, BinaryOperator<T> accumulator);	Stream<T> 전체를 T identity라는 단일 결과값에 Reduction 함
Optional<T> reduce (BinaryOperator<T> accumulator);	Stream<T> 전체를 T 타입의 단일 결과값 Reduction을 가질 수 있도록 반환

reduce



최솟값
1

(e1, e2) → Math.min(e1,e2))



BinaryOperator는 각 쌍에 대해
부분 최소값을 구해냄

Note : Optional은 non-null Stream의 결과로 비어 있는 결과, 비어 있지 않은 결과의 추상화

Note
: accumulator.apply(identity, t) → t

최솟값, 최대값, SUM, MULTIPLY의 identity는?

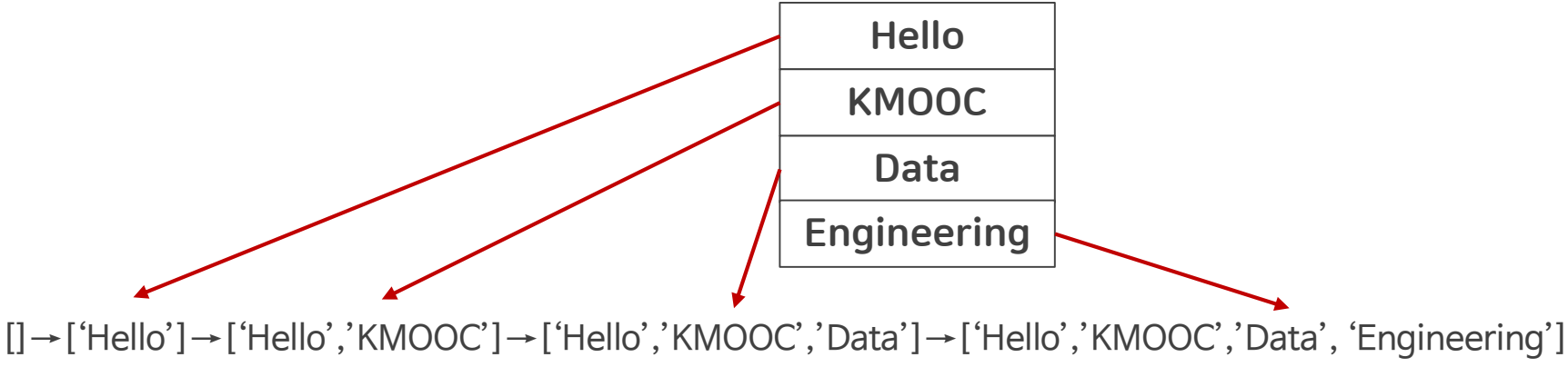
reduce

Stream method	Description
T reduce(T identity, BinaryOperator〈T〉 accumulator);	Stream〈T〉 전체를 T identity라는 단일 결과값에 Reduction 함
Optional〈T〉 reduce (BinaryOperator〈T〉 accumulator);	Stream〈T〉 전체를 T 타입의 단일 결과값 Reduction을 가질 수 있도록 반환

collect

Stream method	Description
⟨R⟩ R collect(Supplier⟨R⟩ supplier, BiConsumer⟨R, ? super T⟩ accumulator, BiConsumer⟨R, R⟩ combiner);	Stream⟨T⟩ 전체에 대한 결과를 담을 용기 supplier를 제공하고, instance를 supplier에 담는 방법 (accumulator)과 병렬 처리시 중간 결과 값들을 병합할 수 있는 combiner를 제공하여 모음

ArrayList⟨String⟩ 용기에 담기



병렬 처리

Supplier

```
String result = Stream.of(1, 2, 3, 4, 5).collect(
() -> new StringBuilder(),

(StringBuilder b, Integer e) -> b.append(String.valueOf(e)),

(StringBuilder b1, StringBuilder b2) -> {}

).toString();
System.out.println(result);
```

병렬처리를 위해서 구현해야 함

Accumulator

built-in Collector

List<E> collect(Collectors.toList())는 어디에?

- 편리를 위한 built-in collector 들을 제공함

Remind

데이터 엔지니어링의 단편을
Java Collection Framework으로 풀어 봄

실 세계 데이터 처리 파이프라인 숙달

문제에 따른 적절한 자료구조 선택 및 활용

Java Collection Framework과
호환 가능한 자료구조 설계 및 구현

Lambda Expression과 Stream 처리를 통한
생산성 향상