Stream

Stream?

다양한 데이터를 표준화된 방법으로 다루기 위한 것이다.

스트림의 특징은 다음과 같다.

- Stream은 원본을 읽기만 할 뿐 변경하지 않는다.
- Stream은 일회용이다.
- Stream은 작업을 내부 반복으로 처리한다.

Stream의 연산

Stream의 연산은 다음과 같다.

- 중간 연산: 연산 결과가 Stream인 연산으로 Stream에 연속해서 중간 연산할 수 있다.
- 최종 연산: 연산 결과가 Stream이 아닌 연산으로 Stream의 요소를 소모하므로 단 한 번만 가능하다.

중간 연산의 종류는 다음과 같다.

중간 연산	설 명
Stream distinct()	중복을 제거한다.
Stream filter(Predicate predicate)	조건에 안 맞는 요소를 제외한다.
Stream limit(long maxSize)	Stream의 일부를 잘라낸다.
Stream skip(long n)	Stream의 일부를 건너뛴다.
Stream peek(Consumer action)	Stream의 요소에 작업을 수행한다.
Stream sorted([Comparator comparator])	Stream의 요소를 정렬한다.
Stream map(Function<T, R> mapper) DoubleStream mapToDouble(ToDoubleFunction mapper) IntStream mapToInt(ToIntFunction mapper) LongStream mapToLong(ToLongFunction mapper) Stream flatMap(Function<T, Stream> mapper) DoubleStream flatMapToDouble(Function<T, DoubleStream> m) IntStream flatMapToInt(Function<T, IntStream> m) LongStream flatMapToLong(Function<T, LongStream> m)	Stream의 요소를 변환한다.

최종 연산의 종류는 다음과 같다.

최종 연산	설 명
void forEach(Consumber<? super T> action) void forEachOrdered(Consumber<? super T> action)	각 요소에 지정된 작업을 수행한다.

최종 연산	설 명
long count()	Stream의 요소에 개수를 반환한다.
Optional max(Comparator<? super T> comparator) Optional min(Comparator<? super T> comparator)	Stream의 최대값/최소값을 반환한다.
Optional findAny() Optional findFirst()	Stream의 요소를 하나 반환한다.
boolean allMatch(Predicate p) boolean anyMatch(Predicate p) boolean noneMatch(Predicate p)	주어진 조건을 요소가 만족시키는지 여부를 확인한다.
Object[] toArray() A[] toArray(IntFunction<A[]> generator)	Stream의 모든 요소를 배열로 반환한다.
Optional reduce(BinaryOperator accumulator) T reduce(T identity, BinaryOperator accumulator) U reduce(U indetity, BiFunction<U, T, U> accumulator, BinaryOperator combiner)	Stream의 요소를 하나씩 줄여가면서 계산한다.
R collect(Collector<T, A, R> collector) R collect(Supplier supplier, BiConsumer<R, T> accumulator, BiConsumber<R, R> combiner)	Stream의 요소를 수집한다.

Stream의 연산의 특징이다.

- 지연된 연산: 최종 연산이 수행되기 전까지는 중간 연산이 수행되지 않는다.
- 기본형 Stream: 오토박싱&언박싱을 생략하여 효율적이다.
- 병렬 Stream: parallel()을 호출하여 연산을 병렬로 수행한다.

Stream의 생성

- 빈 Stream 생성

```
Stream stream_name = Stream.empty();
```

- 컬렉션으로부터 Stream 생성

```
Stream<T> stream_name = Collection.stream();
```

- 배열로부터 Stream 생성

```
Stream<T> stream_name = Stream.of(T[]);
Stream<T> stream_name = Arrays.stream(T[]);
IntStream stream_name = IntStream.of(int... values);
IntStream stream_name = Arrays.stream(int[] array, int start, int end);
```

- 특정 범위의 정수를 요소로 갖는 Stream 생성

```
IntStream stream_name = IntStream.range(int from, int to); // to 포함 X
IntStream stream_name = IntStream.rangeClosed(int from, int to); // to 포함
```

- 임의의 수를 요솔 갖는 Stream 생성

```
IntStream stream_name = new Random().ints(); // 무한 스트림이므로 limit()을 같이 사용
```

- 람다식을 소스로 하는 Stream 생성

```
// 무한 스트림
Stream<T> stream_name = Stream.iterate(T seed, UnaryOperator<T> f); // 이전 요소에 종속적
Stream<T> stream_name = Stream.generate(Supplier<T> s); // 이전 요소에 독립적
```

- 파일을 소스로 하는 Stream 생성

```
Stream<Path> Files.list(Path dir);
```

- 두 Stream을 연결

```
Stream<T> stream_name = Stream.concat(Stream<T> stream1, Stream<T> stream2);
```

Stream의 중간 연산

- Stream 자르기

```
skip(long n) // 처음 n개의 요소를 건너뛴다
limit(long maxSize) // Stream의 요소를 maxSize개로 제한한다
```

- Stream의 요소 걸러내기

```
filter(Predicate<? super T> predicate) // 조건에 맞지 않는 요소를 제거한다
distinct() // 중복을 제거한다
```

- Stream 정렬

```
sorted() // 기본 정렬
/**
* Comparator.comparing() 사용
* 정렬 기준이 여러 개라면 .thenComparing()으로 연결
*/
sorted(Comparator<? super T> comparator) // 지정된 정렬 조건으로 정렬
```

- Stream의 요소 변환

```
map(Function<? super T, ? extends R> mapper)
```

- Stream의 요소 조회

```
peek(Consumber<? super T> action)
```

- Stream을 기본형 Stream으로 변환

```
mapToInt(ToIntFunction<? super T> mapper)
```

- 기본형 Stream을 Stream으로 변환

```
mapToObj(IntFunction<? extends T> mapper)
boxed()
```

- Stream의 Stream을 Stream으로 변환

```
/**
* 여러 개의 배열을 하나의 배열로 바꿀 때 사용
* 여러 줄의 문장들을 단어로 뽑을 때 사용
*/
flatMap()
```

Optional & OptionalInt

Optional는 지네릭스 클래스로 T 타입의 객체를 감싸는 래퍼 클래스이다.
null을 직접 다루면 에러가 발생할 수도 있어 null 체크를 해줘야 하는데, 코드가 지저분해진다.
그럴 때 간접적으로 null 값을 다루는 Optional이다.

Optional 객체 생성

```
Optional<T> opt_name = Optional.of(T value);
Optional<T> opt_name = Optional.ofNullable(T value); // 이거 사용
Optional<T> opt_name = Optional.empty();
```

Optional 객체의 값 가져오기

```
T value = opt_name.get();
T value = opt_name.orElse(T data); // null일 경우 data로 지정
T value = opt_name.orElseGet(Supplier<? extends T> other)
T value = opt_name.orElseThrow(Supplier<? extends X> exceptionSupplier) // null이
면 예외 발생
```

isPresent()는 Optional 객체의 값이 null이면 false를 반환한다.
ifPresent()는 null이 아닐 때만 작업을 수행한다.

기본형 Optional의 값 가져오기

```
OptionalInt intOpt_name = OptionalInt.ofNullable(int value);
int data = intOpt_name.getAsInt();
```

Stream의 최종 연산

- Stream의 모든 요소에 지정된 작업을 수행

```
void forEach(Consumer<? super T> action) // 병렬 Stream인 경우 순서 보장 X
void forEachOrdered(Consumer<? super T> action) // 병렬 Stream인 경우 순서 보
장
```

- Stream을 배열로 변환

```
Object[] toArray()
A[] toArray(IntFunction<A[]> generator)
```

- 조건 검사

```
boolean allMatch(Predicate<? super T> predicate) // 모든 요소가 조건을 만족시키
면 true
boolean anyMatch(Predicate<? super T> predicate) // 한 요소라도 조건을 만족시키
면 true
```

```
boolean noneMatch(Predicate<? super T> predicate) // 모든 요소가 조건을 만족시
키지 않으면 true
// filter()랑 같이 사용
Optional<T> findFirst() // 순차 Stream에 사용하며, 첫 번째 요소를 반환
Optional<T> findAny() // 병렬 Stream에 사용하며, 아무거나 하나를 반환
```

- 통계

```
long count()
Optional<T> max(Comparator<? super T> comparator)
Optional<T> min(Comparator<? super T> comparator)
```

- Stream의 요소를 하나씩 줄여가며 누적 연산 수행

```
/**
* 전체를 대상으로 연산할 때 사용
* identity 초기값
* accumulator 연산
* combiner 병렬 Stream
*/
Optional<T> reduce(BinaryOperator<T> accumulator)
T reduce(T identity, BinaryOperator<T> accumulator)
U reduce(U identity, BiFunction<U, T, U> accumulator, BinaryOperator<U>
combiner)
```

collect()

reduce()가 전체를 대상으로 했다면 collect()는 그룹별로 연산을 한다.
collect()는 Stream의 최종 연산으로, 매개변수로 인터페이스인 Collector를 필요로 한다.
Collectors는 static 메서드로 미리 작성된 Collector를 제공하는 클래스이다.

```
Object collect(Collector collector)
Object collect(Supplier supplier, BiConsumer accumulator, BiConsumer combiner)
```

Collectors의 메서드

- Stream을 컬렉션과 배열로 변환

```
List toList()
Set toSet()
Map toMap()
Collection toCollection() // 구현 클래스를 지정할 때 사용
T[] toArray()
```

- Stream의 통계

```
long counting()
int summingInt()
int averagingInt()
OpitonalInt maxBy()
OpitonalInt minBy()
```

- Stream의 누적 연산

```
Collector reducing(BinaryOperator<T> accumulator)
Collector reducing(T identity, BinaryOperator<T> accumulator)
Collector reducing(U identity, BiFunction<U, T, U> accumulator,
BinaryOperator<U> combiner)
```

- 문자열 Stream의 요소를 모두 연결

```
Collector joining()
```

- 그룹화와 분할

```
// n분할
Collector groupingBy(Function classifier)
Collector groupingBy(Function classifier, Collector downstream)
Collector groupingBy(Function classifier, Supplier mapFactory, Collector
downstream)
// 2분할
Collector partitioningBy(Predicate predicate)
Collector partitioningBy(Predicate predicate, Collector downstream)
```

Collector 구현

```
public interface Collector<T, A, R> {
    Supplier<A> supplier(); // 결과를 저장할 공간을 제공
    BiConsumer<A, T> accumulator(); // Stream의 요소를 수집할 방법을 제공
    BinaryOperator<A> combiner(); // 두 저장 공간을 병합할 방법을 제공(병렬 Stream)
    Function<A, R> finisher(); // 최종 변환
    Set<Characteristics> characteristics(); // Collector의 특성이 담긴 Set을 반환
    /**
    * Collector가 수행할 작업의 속성 정보를 제공
    * Characteristics.CONCURRENT // 병렬로 처리할 수 있는 작업
    * Characteristics.UNORDERED // Stream의 요소의 순서가 유지될 필욧가 없는 작업
    * Characteristics.IDENTITY_FINISH // finisher()가 항등 함수인 작업
```

```
    * 속성이 없는 경우 return Collections.emptySet();
    */
}
```