yonggyo1125 Update README.md 9809671 · last year History

Name	Name	Last commit date
..		
README.md	Update README.md	last year

README.md

강의 동영상 링크

동영상 링크

스트림(Stream)

스트림이란?

- 데이터 소스가 무엇이든 간에 같은 방식으로 다룰 수 있게 데이터를 추상화 하고 데이터를 다루는데 자주 사용되는 메서드들을 정의해 놓음
- 데이터 소스가 무엇이든 간에 같은 방식으로 다룰 수 있게 되면서 코드의 재사용성이 높아진다.
- 배열이나, 컬렉션뿐만 아니라 파일에 저장된 데이터도 모두 같은 방식으로 다룰 수 있다.

스트림은 데이터 소스를 변경하지 않는다.

- 스트림은 데이터 소스로 부터 데이터를 읽기만할 뿐, 데이터 소스를 변경하지 않는다.

스트림은 일회용이다.

- 스트림은 한번 사용하면 닫혀서 다시 사용할 수 없다.
- 다시 사용하려면 스트림을 다시 생성해야 한다.

```
strStream1.sorted().forEach(x -> System.out.println(x));
```

```
int numOfStr = strStream1.count(); // 스트림이 닫혔으므로 에러 발생
```

스트림은 작업을 내부 반복으로 처리한다.

- 내부 반복이라는 것은 반복문을 메서드 내부에 숨길 수 있다는 것을 의미

```
void forEach(Consumer<? super T> action) {
        Objects.requireNonNull(action); // 매개변수의 널 체크

        for(T t : src) {
                action.accept(T);
        }
}
```

스트림의 연산

중간연산

연산 결과가 스트림인 연산, 스트림에 연속해서 중간 연산할 수 있다.

최종연산

연산 결과가 스트림이 아닌 연산. 스트림의 요소를 소모하므로 단 한번만 가능

```
stream.distinct().limit(5).sorted().forEach(x -> System.out.println(x));

distinct(), limit(5), sorted() - 중간연산
forEach - 최종연산
```

스트림의 중간 연산 목록

중간 연산	설명
Stream<T> distinct()	중복을 제거
Stream<T> filter(Predicate<T> predicate)	조건에 안 맞는 요소 제외
Stream<T> limit(long maxSize)	스트림의 일부를 잘라낸다
Stream<T> skip(long n)	스트림의 일부를 건너뛴다
Stream<T> peek(Consumer<T> action)	스트림의 요소에 작업 수행
Stream<T> sorted() Stream<T> sorted(Comparator<T> comparator)	스트림의 요소를 정렬한다.
Stream<R> map(Function<T,R> mapper) DoubleStream mapToDouble(ToDoubleFunction<T> mapper) IntStream mapToInt(ToIntFunction<T> mapper)	스트림의 요소를 변환한다.

중간 연산	설명
Stream<R> flatMap(Function<T, Stream<R>> mapper) DoubleStream flatMapToDouble(Function<T, DoubleStream> m) IntStream flatMapToInt(Function<T, IntStream> m) LongStream flatMapToLong(Function<T, LongStream> m)	

스트림의 최종 연산 목록

최종 연산	설명
void forEach(Consumer<? super T> action)	void forEachOrdered(Consumer<? super T> action)
long count())	스트림의 요소의 개수 반환
Optional<T> max(Comparator<? super T> comparator) Optional<T> min(Comparator<? super T> comparator)	스트림의 최대값/최소값을 반환
Optional<T> findAny() // 아무거나 하나 Optional<T> findFirst() // 첫 번째 요소	스트림의 요소를 하나 반환

boolean allMatch(Predicate<T> p) // 모두 만족하는지
boolean anyMatch(Predicate<T> p) // 하나라도 만족하는지
boolean noneMatch(Predicate<T> p) // 모두 만족하지 않는지
	주어진 조건을 모든 요소가 만족시키는지, 만족시키지 않는지 확인

Object[] toArray() A[] toArray(IntFunction<A[]> generator)	스트림의 모든 요소를 배열로 변환
Optional<T> reduce(BinaryOperator<T> accumulator) T reduce(T identity, BinaryOperator<T> accumulator)	스트림 요소를 하나씩 줄여가면서(리듀싱) 계산한다.
R collect(Collector<T,A,B> collector)	스트림의 요소를 수집한다. 주로 요소를 그룹화하거나 분할한 결과를 컬렉션에 담아 반환하는데 사용한다.

지연된연산

- 최종 연산이 수행되기 전까지는 중간 연산이 수행되지 않는다.
- 중간 연산을 호출하는 것은 단순히 어떤 작업이 수행되어야하는 지를 지정해주는 것이다.
- 최종 연산이 수행되어야 스트림의 요소들이 중간 연산을 거쳐 최종 연산에서 소모된다.

기본자료형을 다루는 스트림

- 요소의 타입이 T인 스트림은 기본적으로 Stream인데, 기본 자료형을 다루려면 오토박싱&언박싱이 발생하여 비효율성이 증가한다(예 - Integer <-> int)
- 비효율성을 줄이기 위해 데이터 소스의 요소를 기본형으로 다루는 스트림이 제공된다.
- IntStream, LongStream, DoubleStream
- 기본자료형에 유용하게 사용할 수 있는 메서드를 제공한다.

스트림만들기

컬렉션

```
Stream<T> Collection.stream()
```

```
List<Integer> list = Arrays.asList(1,2,3,4,5);
Stream<Integer> intStream = list.stream();

intStream.forEach(i -> System.out.println(i));  // 또는 메서드 참조 방식으로 람다식 정의 intStream.forEach(System.out::println);
```

배열

```
Stream<T> Stream.of(T... values) // 가변인자
Stream<T> Stream.of(T[])
Stream<T> Arrays.stream(T[])
Stream<T> Arrays.stream(T[] array, int startInclusive, int endExclusive)  // startInclusive이상 endExclusive 미만 범위의 스트림 생성
```

```
사용 예)
Stream<String> strStream = Stream.of("a", "b", "c"); // 가변인자
Stream<String> strStream = Stream.of(new String[] {"a", "b", "c"});
Stream<String> strStream = Arrays.stream(new String[]{"a", "b","c"});
Stream<String> strStream = Arrays.stream(new String[]{"a", "b", "c", "d"}, 0, 3);
```

int, long, double과 같은 기본형 배열을 소스로 하는 스트림을 생성하는 메서드

```
IntStream IntStream.of(int... values)
IntStream IntStream.of(int[])
IntStream Arrays.stream(int[])
IntStream Arrays.stream(int[] array, int startInclusive, inbt endExclusive) // startInclusive이상 endExclusive 미만 범위의 스트림 생성
```

특정 범위의 정수

```
IntStream IntStream.range(int begin, int end)  // begin이상 ~ end 미만
IntStream IntStream.rangeClosed(int begin, int end) // begin 이상 ~ end 이하
```

임의의 수

Random 클래스에는 난수들로 이루어진 스트림을 반환하는 메서드를 가지고 있다.

```
IntStream ints()
LongStream longs()
DoubleStream doubles()
```

- 매개변수가 없다면 크키가 정해지지 않은 무한 스트림
- limit()를 함께 사용해서 스트림의 크기를 제한해 줘야 한다.

```
IntStream intStream = new Random().ints(); // 무한 스트림
intStream.limit(5).forEach(System.out::println); // 5개의 요소만 출력
```

- 매개변수가 있다면 크기가 정해진 유한스트림, limit()를 사용하지 않아도 된다.

```
IntStream ints(long streamSize)
LongStream longs(long streamSize)
DoubleStream doubles(long streamSize)
```

```
IntStream intStream = new Random().ints(5); // 크키가 5인 난수 스트림 반환
```

- 지정된 범위의 난수를 발생시키는 스트림을 얻는 메서드

```
begin이상 end 미만 범위에서 난수 발생

IntStream  ints(int begin, int end)
LongStream longs(long begin, long end)
DoubleStream doubles(double begin, double end)

IntStream ints(long streamSize, int begin, int end)
LongStream longs(long streamSize, long begin, long end)
DoubleStream doubles(long streamSize, double begin, double end)
```

람다식 - iterate(), generate()

람다식을 매개변수로 받아서 람다식에 의해 계산되는 값들을 요소로 하는 무한 스트림을 생성

```
static <T> Stream<T> iterate(T seed, UnaryOperator<T> f)
static<T> Stream<T> generate(Supplier<T> s)
```

- iterate()는 이전 결과를 이용해서 다음요소를 계산

```
Stream<Integer> evenStream = Stream.iterate(0, n->n+2); // 0, 2, 4, 6 ...
```

- generate()는 이전 결과를 사용하지 않고 람다식에 의해 계산되는 값을 요소로 하는 무한 스트림을 생성하여 반환
- generate()는 이전 결과를 사용하지 않으므로 매개변수 타입이 Suppiler 이다.

```
Stream<Double> randomStream = Stream.generate(Math::random);
Stream<Integer> oneStream = Stream.generate(() -> 1);
```

빈 스트림

- 요소가 하나도 없는 비어있는 스트림
- 스트림에서 연산을 수행한 결과가 하나도 없을 때, null 보다 빈 스트림을 반환하는 것이 좋다.

```
Stream emptyStream = Stream.empty(); // 빈 스트림을 생성해서 반환
long count = emptyStream.count(); // count의 값은 0
```

두 스트림의 연결 - concat()

```
String[] str1 = {"123", "456", "789"}
String[] str2 = {"ABC", "abc", "DEF"};

Stream<String> strs1 = Stream.of(str1);
Stream<String> strs2 = Stream.of(str2);
Stream<String> strs3 = Stream.concat(strs1, strs2);
```

스트림의 중간 연산

스트림 자르기 - skip(), limit()

```
Stream<T> skip(long n)   // n만큼 건너뛰기
Stream<T> limit(long maxSize) // maxSize 만큼 자르기
```

```
IntStream intStream = IntStream.rangeClosed(1, 10); // 1~10의 요소를 가진 스트림
intStream.skip(3).limit(5).forEach(System.out::println); // 45678
```

스트림 요소 걸러내기 - filter(), distinct()

- filter() - 주어진 조건(Predicate)에 맞지 않는 요소를 걸러낸다.
- distinct() - 스트림에서 중복된 요소를 제거

```
Stream<T> filter(Predicate<? super T> predicate)
Stream<T> distinct()
```

- distinct()

```
IntStream intStream = IntStream.of(1, 2, 2, 3, 3, 4, 5, 5, 6);
intStream.distinct().forEach(System.out::print); // 123456
```

- filter()

```
IntStream intStream = IntStream.rangeClosed(1, 10); // 1 ~ 10
IntStream.filter( i -> i % 2 == 0).forEach(System.out::print); // 246810
```

정렬 - sorted()

```
Stream<T> sorted()
Stream<T> sorted(Comparator<? super T> comparator)
```

Comparator를 지정하지 않으면 스트림 요소의 기본 정렬 기준(Comparable)으로 정렬한다. 단, 스트림의 요소가 Comparable을 구현한 클래스가 아니면 예외가 발생한다.

```
Stream<String> strStream = Stream.of("dd", "aaa", "CC", "cc", "b");
strStream.sort().forEach(System.out::println);
```

day19/Student.java

```
package day19;

public class Student implements Comparable<Student> {
        String name;
        int ban;
        int totalScore;
        Student(String name, int ban, int totalScore) {
                this.name = name;
                this.ban = ban;
                this.totalScore = totalScore;
        }

        public String toString() {
                return String.format("[%s, %d, %d]", name, ban, totalScore).toString(
        }

        String getName() { return name; }
        int getBan() { return ban; }
        int getTotalScore() { return totalScore; }

        // 총점 내림차순을 기본 정렬로 한다.
```

```
        public int compareTo(Student s) {
                return s.totalScore - this.totalScore;
        }
}
```

day19/StreamEx1.java

```
package day19;

import java.util.*;
import java.util.stream.*;

public class StreamEx1 {

        public static void main(String[] args) {
                Stream<Student> studentStream = Stream.of(
                                        new Student("이자바", 3, 300),
                                        new Student("김자바", 1, 200),
                                        new Student("안자바", 2, 100),
                                        new Student("박자바", 2, 150),
                                        new Student("소자바", 1, 200),
                                        new Student("나자바", 3, 290),
                                        new Student("강자바", 3, 180)
                                );

                studentStream.sorted(Comparator.comparing(Student::getBan) // 반별 정
                                        .thenComparing(Comparator.naturalOrder())) //
                                        .forEach(System.out::println);
        }
}
```

- 실행결과

```
[김자바, 1, 200]
[소자바, 1, 200]
[박자바, 2, 150]
[안자바, 2, 100]
[이자바, 3, 300]
[나자바, 3, 290]
[강자바, 3, 180]
```

변환 - map()

스트림의 요소에서 저장된 값 중에서 원하는 필드만 뽑아내거나 특정 형태로 변환해야 하는 경우

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

```
Stream<File> fileStream = Stream.of(new File("Ex1.java"), new File("Ex1"), new
File("Ex1.bak"),
new File("Ex2.java"), new File("Ex1.txt"));

// map()으로 Stream<File>을 Stream<String>으로 변환
Stream<String> filenameStream = fileStream.map(File::getName);
filenameStream.forEach(System.out::println); // 스트림의 모든 파일이름을 출력
```

조회 - peek()

- forEach와 비슷하나 스트림의 요소를 소모하지 않는 중간 연산
- 중간연산이므로 연산 사이에 여러번 넣어도 된다.

day19/StreamEx2.java

```
package day19;

import java.io.*;
import java.util.stream.*;

public class StreamEx2 {

	public static void main(String[] args) {
		File[] fileArr = { new File("Ex1.java"), new File("Ex1.bak"),
				new File("Ex2.java"), new File("Ex1"), new File("Ex1.

		Stream<File> fileStream = Stream.of(fileArr);

		// map()으로 Stream<File>을 Stream<String>으로 변환
		Stream<String> filenameStream = fileStream.map(File::getName);
		filenameStream.forEach(System.out::println); // 모든 파일의 이름을 출ㄹ

		fileStream = Stream.of(fileArr); // 스트림을 다시 생성

		fileStream.map(File::getName)
			.filter(s -> s.indexOf('.') != -1) // 확장자가 없는 것은 제외
			.map(s -> s.substring(s.indexOf('.') + 1)) // 확장자만 추출
			.map(String::toUpperCase) // 모두 대문자로 변환
			.distinct() // 중복 제거
			.forEach(System.out::print);  // JAVABAKTXT
	}
}
```

- 실행결과

```
package day19;

import java.io.*;
import java.util.stream.*;

public class StreamEx2 {
```

```
        public static void main(String[] args) {
                File[] fileArr = { new File("Ex1.java"), new File("Ex1.bak"),
                                new File("Ex2.java"), new File("Ex1"), new
File("Ex1.txt")};

                Stream<File> fileStream = Stream.of(fileArr);

                // map()으로 Stream<File>을 Stream<String>으로 변환
                Stream<String> filenameStream = fileStream.map(File::getName);
                filenameStream.forEach(System.out::println); // 모든 파일의 이름을
출력

                fileStream = Stream.of(fileArr); // 스트림을 다시 생성

                fileStream.map(File::getName)
                        .filter(s -> s.indexOf('.') != -1) // 확장자가 없는 것은 제
외
                        .map(s -> s.substring(s.indexOf('.') + 1)) // 확장자만 추출
                        .map(String::toUpperCase) // 모두 대문자로 변환
                        .distinct() // 중복 제거
                        .forEach(System.out::print);  // JAVABAKTXT
        }
}
```

mapToInt(), mapToLong(), mapToDouble()

map()은 연산결과로 Stream 타입입의 스트림을 반환하지만 기본자료형인 int, long, double으로 반환해 주는 기본 스트림을 반환

```
DoubleStream mapToDouble(ToDoubleFunction<? super T> mapper)
IntStream mapToInt(ToIntFunction<? super T> mapper)
LongStream mapToLong(ToLongFunction<? super T> mapper)
```

```
IntStream studentScoreStream = studentStream.mapToInt(Student::getTotalScore);
int allTotalScore = studentScoreStream.sum();
```

참고) 기본형 스트림은 숫자를 다루는 편리한 메서드를 제공

메서드	메서드 설명
int sum()	스트림의 모든 요소의 총합
OptionalDouble average()	스트림 요소의 평균
OptionalInt max()	스트림 요소 중 제일 큰 값
OptionalInt min()	스트림 요소 중 제일 작은 값
IntSummaryStatistics summaryStatistics()	스트림의 통계 요약 정보

day19/StreamEx3.java

```java
package day19;

import java.util.*;
import java.util.stream.*;

public class StreamEx3 {
	public static void main(String[] args) {
		Student[] stuArr = {
				new Student("이자바", 3, 300),
				new Student("김자바", 1, 200),
				new Student("안자바", 2, 100),
				new Student("박자바", 2, 150),
				new Student("소자바", 1, 200),
				new Student("나자바", 3, 290),
				new Student("강자바", 3, 180)
		};

		Stream<Student> stuStream = Stream.of(stuArr);

		stuStream.sorted(Comparator.comparing(Student::getBan)
				.thenComparing(Comparator.naturalOrder()))
				.forEach(System.out::println);

		stuStream = Stream.of(stuArr); // 스트림을 다시 생성한다.
		IntStream stuScoreStream = stuStream.mapToInt(Student::getTotalScore)

		IntSummaryStatistics stat = stuScoreStream.summaryStatistics();
		System.out.println("count=" + stat.getCount());
		System.out.println("sum=" + stat.getSum());
		System.out.printf("average=%.2f%n", stat.getAverage());
		System.out.println("min=" + stat.getMin());
		System.out.println("max=" + stat.getMax());
	}

}
```

- 실행결과

```
[김자바, 1, 200]
[소자바, 1, 200]
[박자바, 2, 150]
[안자바, 2, 100]
[이자바, 3, 300]
[나자바, 3, 290]
[강자바, 3, 180]
count=7
sum=1420
average=202.86
min=100
max=300
```

flatMap() - Stream<T[]>를 Stream로 변환

스트림의 타입이 Stream<T[]>인 경우 Stream로 변환해 준다.

```
Stream<String> strStrm = Stream.of("abc", "def", "jklmn");
Stream<String> strStrm2 = Stream.of("ABC", "GHI", "JKLMN");

Stream<Stream<String>> strmStrm = Stream.of(strStrm, strStrm2);
Stream<String> strStream = strmStrm.map(s -> s.toArray(String[]::new)) //
Stream<Stream<String>> -> Stream<String[]>

.flatMap(Arrays::stream); // Stream<String[]> -> Stream<String>
```

day19/StreamEx4.java

```
package day19;

import java.util.*;
import java.util.stream.*;

public class StreamEx4 {
	public static void main(String[] args) {
		Stream<String[]> strArrStrm = Stream.of(
				new String[] {"abc", "def", "jkl"},
				new String[] {"ABC", "GHI", "JKL"}
		);
		// Stream<Stream<String>> strStrmStrm = strArrStrm.map(Arrays::stream
		Stream<String> strStrm = strArrStrm.flatMap(Arrays::stream);

		strStrm.map(String::toLowerCase)
			.distinct()
			.sorted()
			.forEach(System.out::println);
		System.out.println();

		String[] lineArr = {
				"Believe or not It is true",
				"Do or do not There is no try",
		};

		Stream<String> lineStream = Arrays.stream(lineArr);
		lineStream.flatMap(line -> Stream.of(line.split(" +")))
			.map(String::toLowerCase)
			.distinct()
			.sorted()
			.forEach(System.out::println);
		System.out.println();

		Stream<String> strStrm1 = Stream.of("AAA", "ABC", "bBb", "Dd");
		Stream<String> strStrm2 = Stream.of("bbb", "aaa", "ccc", "dd");

		Stream<Stream<String>> strStrmStrm = Stream.of(strStrm1, strStrm2);
		Stream<String> strStream = strStrmStrm
					.map(s -> s.toArray(String[]::new))
					.flatMap(Arrays::stream);
		strStream.map(String::toLowerCase)
```

```
                        .distinct()
                        .forEach(System.out::println);
        }
}
```

- 실행결과

```
abc
def
ghi
jkl

believe
do
is
it
no
not
or
there
true
try

aaa
abc
bbb
dd
ccc
```

Optional와 OptionalInt

- Optional은 제네릭 클래스로 "T타입의 객체"를 감싸는 래퍼 클래스이다.
- Optional 타입의 객체에는 모든 타입의 참조변수를 담을 수 있다.

```
public final class Optional<T> {
        private final T value;  // T 타입의 참조변수
        ...
}
```

- 스트림에서 최종연산의 결과를 Optional 객체에 담아서 반환
- 객체에 담아서 반환을 하므로 결과가 null인지를 간단하게 체크하는 메서드를 제공한다.
- if로 따로 체크하지 않아도 NullPointException이 발생하지 않는 보다 간결하고 안전한 코드 작성이 가능해 진다.

Optional 객체 생성하기

- of() 또는 ofNullable()을 사용
- 참조변수의 값이 null일 가능성이 있으면 of()대신 ofNullable()을 사용해야 한다.

- Optional타입의 참조 변수를 기본값으로 초기화 할때는 empty()를 사용한다.

```
String str = "abc";
Optional<String> optVal = Optional.of(str);
Optional<String> optVal = Optional.of("abc");
Optional<String> optVal = Optional.of(new String("abc"));
```

```
Optional<String> optVal = Optional.of(null); // NullPointException 발생
Optional<String> optVal = Optional.ofNullable(null);
```

```
Optional<String> optVal = Optional.<String>empty(); // 빈 객체로 초기화
```

Optional 객체의 값 가져오기

- get()을 사용하여 Optional 객체에 저장된 값을 가져온다. 값이 null일 때는 NoSuchElementException이 발생한다.
- orElse("기본값")을 사용하면 값이 null일때 "기본값"으로 대체할 수 있다.
- 람다식을 매개변수로 하여 저장된 값을 가져올 수 있다.

```
T orElseGet(Supplier<? extends T> other)
T orElseThrow(Supplier<? extends X> exceptionSupplier)
```

```
String str3 = optVal2.orElseGet(String::new);
String str4 = optVal2.orElseThrow(NullPointerException::new);
```

OptionalInt, OptionalLong, OptionalDouble

- IntStream과 같은 기본형 스트림에는 Optional도 기본형을 값으로 하는 OptionalInt, OptionalLong, OptionalDouble을 반환한다.
- IntStream에 정의된 메서드 예

```
OptionalInt findAny()
OptionalInt findFirst()
OptionalInt reduce(IntBinaryOperator op)
OptionalInt max()
OptionalInt min()
OptionalDouble average()
```

- 반환타입이 Optional가 아니라는 것을 제외하고는 Stream에 정의된 것과 비슷하나 Optional에 저장된 값을 꺼낼 때 사용하는 메서드의 이름이 조금씩 다르다

Optinal클래스	값을 반환하는 메서드
Optional	T get()

Optinal클래스	값을 반환하는 메서드
OptionalInt	int getAsInt()
OptionalLong	long getAsLong()
OptionalDouble	double getAsDouble()

day19/OptionalEx1.java

```
package day19;

import java.util.*;

public class OptionalEx1 {
	public static void main(String[] args) {
		Optional<String> optStr = Optional.of("abcde");
		Optional<Integer> optInt = optStr.map(String::length);
		System.out.println("optStr=" + optStr.get());
		System.out.println("optInt=" + optInt.get());

		int result1 = Optional.of("123")
					.filter(x->x.length() > 0)
					.map(Integer::parseInt).get();

		int result2 = Optional.of("")
					.filter(x->x.length() > 0)
					.map(Integer::parseInt).orElse(-1);

		System.out.println("result1=" + result1);
		System.out.println("result2=" + result2);

		Optional.of("456").map(Integer::parseInt)
				.ifPresent(x->System.out.printf("result3=%d%n", x));

		OptionalInt optInt1 = OptionalInt.of(0); // 0을 저장
		OptionalInt optInt2 = OptionalInt.empty(); // 빈 객체를 생성

		System.out.println(optInt1.isPresent()); // true
		System.out.println(optInt2.isPresent()); // false;

		System.out.println(optInt1.getAsInt()); // 0
		// System.out.println(optInt2.getAsInt()); // NoSuchElementException
		System.out.println("optInt1=" + optInt1);
		System.out.println("optInt2=" + optInt2);
		System.out.println("optInt1.equals(optInt2)?" + optInt1.equals(optInt

		Optional<String> opt = Optional.ofNullable(null); // null을 저장
		Optional<String> opt2 = Optional.empty(); // 빈 객체를 생성
		System.out.println("opt = " + opt);
		System.out.println("opt2 = " + opt2);
		System.out.println("opt.equals(opt2)?" + opt.equals(opt2)); // true

		int result3 = optStrToInt(Optional.of("123"), 0);
		int result4 = optStrToInt(Optional.of(""), 0);
```

```
                System.out.println("result3=" + result3);
                System.out.println("result4=" + result4);
        }

        static int optStrToInt(Optional<String> optStr, int defaultValue) {
                try {
                        return optStr.map(Integer::parseInt).get();
                } catch (Exception e) {
                        return defaultValue;
                }
        }
}
```

- 실행결과

```
optStr=abcde
optInt=5
result1=123
result2=-1
result3=456
true
false
0
optInt1=OptionalInt[0]
optInt2=OptionalInt.empty
optInt1.equals(optInt2)?false
opt = Optional.empty
opt2 = Optional.empty
opt.equals(opt2)?true
result3=123
result4=0
```

스트림의 최종 연산

- 최종 연산은 스트림의 요소를 소모해서 결과를
- 최종 연산후에는 스트림이 닫히게 되고 더 이상 사용할 수 없다.

forEach()

- forEach()는 peek()와 달리 스트림의 요소를 소모하는 최종연산이다
- 반환 타입이 void이므로 스트림의 요소를 출력하는 용도로 사용된다.

```
void forEach(Consumer<? super T> action)
```

조건 검사 - allMatch(), anyMatch(), noneMatch(), findFirst(), findAny()

- allMatch() : 지정된 조건에 모든 요소가 일치하는지

- anyMatch() : 지정된 조건에 일부 요소가 일치하는지
- noneMatch() : 지정된 조건에 어떤 요소도 일치하지 않는지
- findFirst() : 지정된 조건에 일치하는 첫 번째 것을 반환
- findAny() : 지정된 조건에 일치하는 첫 번째 것을 반환(병렬 스트림에서 사용)

```
boolean allMatch(Predicate<? super T> predicate)
boolean anyMatch(Predicate<? super T> predicate)
boolean noneMatch(Predicate<? super T> predicate)
```

```
boolean noFailed = stuStream.anyMatch(s->s.getTotalScore() <= 100);
```

```
Optional<Student> stu = stuStream.filter(s->s.getTotalScore() <= 10).findFirst();
```

통계 - count(), sum(), average(), max(), min()

- IntStream과 같은 기본형 스트림에는 스트림의 요소들에 대한 통계 정보를 얻을 수 있는 메서드들이 있다.
- 기본형 스트림이 아닌 경우에는 통계 관련 메서드가 3개만 있다(count(), max(), min())

리듀싱 - reduce()

- 스트림의 요소를 줄여나가면서 연산을 수행하고 최종결과를 반환한다.
- 매개변수의 타입은 BinaryOperator이다.
- 처음 두 요소를 가지고 연산한 결과를 가지고 그 다음 요소를 연산한다.

```
Optional<T> reduce(BinaryOperator<T> accumulator)
T reduce(T identity, BinaryOperator<T> accumulator)
```

참고 BinaryOperator는 BiFunction의 하위 인터페이스이며 BiFunction<T,T,T>와 동등하다.

- 최종연산 count(), sum(), max(), min() 등은 내부적으로 모두 reduce()를 이용해서 작성된 것이다.

```
int count = intStream.reduce(0, (a, b) -> a + 1);
int sum = intStream.reduce(0, (a,b) -> a + b);
int max = intStream.reduce(Integer.MIN_VALUE, (a, b) -> a>b?a:b);
int min = intStream.reduce(Integer.MAX_VALUE, (a, b) -> a<b?a:b);
```

day19/StreamEx5.java

```
package day19;

import java.util.*;
import java.util.stream.*;
```

```
public class StreamEx5 {
	public static void main(String[] args) {
		String[] strArr = {
			"Inheritance", "Java", "Lambda", "stream",
			"optionalDouble", "IntStream", "count", "sum"
		};
		Stream.of(strArr).forEach(System.out::println);

		boolean noEmptyStr = Stream.of(strArr).noneMatch(s->s.length() == 0);
		Optional<String> sWord = Stream.of(strArr)
					.filter(s->s.charAt(0) == 's').findFirst();

		System.out.println("noEmptyStr=" + noEmptyStr);
		System.out.println("sWord=" + sWord.get());

		// Stream<String[]>을 IntStream으로 변환
		IntStream intStream1 = Stream.of(strArr).mapToInt(String::length);
		IntStream intStream2 = Stream.of(strArr).mapToInt(String::length);
		IntStream intStream3 = Stream.of(strArr).mapToInt(String::length);
		IntStream intStream4 = Stream.of(strArr).mapToInt(String::length);

		int count = intStream1.reduce(0, (a,b) -> a + 1);
		int sum = intStream2.reduce(0,  (a,b) -> a + b);

		OptionalInt max = intStream3.reduce(Integer::max);
		OptionalInt min = intStream4.reduce(Integer::min);

		System.out.println("count=" + count);
		System.out.println("sum=" + sum);
		System.out.println("max=" + max.getAsInt());
		System.out.println("min=" + min.getAsInt());
	}
}
```

- 실행결과

```
Inheritance
Java
Lambda
stream
optionalDouble
IntStream
count
sum
noEmptyStr=true
sWord=stream
count=8
sum=58
max=14
min=3
```

collect()

- 스트림의 최종연산, 매개변수로 컬렉터를 필요로 한다.

```
Object collect(Collector collector) // Collector를 구현한 클래스의 객체를 매개변수로
```

```
collect() 스트림의 최종연산, 매개변수로 컬렉터를 필요로 한다.
Collector 인터페이스, 컬렉터는 이 인터페이스를 구현해야 한다.
Collectors 클래스, static 메서드로 미리 작성된 컬렉터를 제공한다.
```

스트림 컬렉션과 배열로 변환 - toList(), toSet(), toMap(), toCollection(), toArray()

- 스트림의 모든 요소를 컬렉션에 수집하고자 할때는 Collectors클래스의 toList()와 같은 메서드를 사용한다.
- List나 Set이 아닌 특정 컬렉션을 지정하려면 toCollection()에 해당하는 컬렉션의 생성자 참조를 매개변수로 넣어준다.

```
List<String> names = stuStream.map(Student::getName).collect(Collectors.toList());
ArrayList<String> list =
names.stream().collect(Collectors.toCollection(ArrayList::new));
```

- toMap()

```
Map<String, Person> map = personStream.collect(Collectors.toMap(p->p.getRegId(),
p->p));
Map<String, Person> map = personStream.collect(Collectors.toMap(p->p.getRegId(),
Function.identity()));
```

- toArray()
  - 매개변수로 해당타입의 생성자를 지정해야 한다.
  - 매개변수를 지정하지 않으면 반환되는 배열의 타입은 'Object[]'이다.

```
Student[] stuNames = studentStream.toArray(Student[]::new);  // OK
Student[] stuNames = studentStream.toArray(); // 에러 발생
Object[] stuNames = studentStream.toArray(); // OK
```

통계 - counting(), summingInt(), averagingInt(), maxBy(), minBy()

```
long count = stuStream.count();
long count = stuStream.collect(Collectors.counting());

long totalScore = stuStream.mapToInt(Student::getTotalScore).sum();
long totalScore =
stuStream.collect(Collectors.summingInt(Student::getTotalScore));
```

```
OptionalInt toScore = studentStream.mapToInt(Student::getTotalScore).max();
Optional<Student> topStudent =
stuStream.max(Comparator.comparingInt(Student::getTotalScore));
Optional<Student> topStudent =
stuStream.collect(maxBy(Comparator.comparingInt(Student::getTotalScore)));

IntSummaryStatistics stat =
stuStream.mapToInt(Student::getTotalScore).summaryStatistics();
IntSummaryStatistics stat =
stuStream.collect(summarizingInt(Student::getTotalScore));
```

리듀싱 - reducing()

```
Collector reducing(BinaryOperator<T> op)
Collector reducing(T identity, BinaryOperator<T> op)
Collector reducing(U identity, Function<T, U> mapper, BinaryOperator<U> op)
```

```
IntStream intStream = new Random().ints(1, 46).distinct().limit(6);

OptionalInt max = intStream.reduce(Integer::max);
Optional<Integer> max =
intStream.boxed().collect(Collectors.reducing(Integer::max));

long sum = intStream.reduce(0, (a, b) -> a + b);
long sum = intStream.boxed().collect(Collectors.reducing(0,
Student::getTotalScore, Integer::sum)));
```

IntStream에는 매개변수 3개짜리 collect()만 정의되어 있으므로 매개변수 1개짜리 collect()를 쓰려면 boxed()를 통해 IntStream을 Stream로 변환해야 한다.

문자열 결합 - joining()

- 문자열 스트림의 모든 요소를 하나의 문자열로 연결해서 반한다.
- 구분자를 지정할 수 있고, 접두사와 접미사도 지정가능하다.
- 스트림의 요소가 String이나 StringBuffer 처럼 CharSequence의 하위 인터페이스인 경우에만 결합이 가능하다
- 스트림의 요소가 문자열이 아닌 경우에는 먼저 map()을 이용해서 스트림의 요소를 문자열로 변환해야 한다.

```
String studentNames =
stuStream.map(Student::getName).collect(Collectors.joining());
String studentNames =
stuStream.map(Student::getName).collect(Collectors.joining(","));
String studentNames =
stuStream.map(Student::getName).collect(Collectors.joining(",","[", "]"));

// map() 없이 스트림에 바로 joining()하면, 스트림의 요소에 toString()을 호출한 결과를
결합
```

```
String studentInfo = studentStream.collect(Collectors.joining(",")); // Student의
toString()으로 결합
```

day19/StreamEx6.java

```
package day19;

import java.util.*;
import java.util.stream.*;
import static java.util.stream.Collectors.*;

public class StreamEx6 {
        public static void main(String[] args) {
                Student[] stuArr = {
                                new Student("이자바", 3, 300),
                                new Student("김자바", 1, 200),
                                new Student("안자바", 2, 100),
                                new Student("박자바", 2, 150),
                                new Student("소자바", 1, 200),
                                new Student("나자바", 3, 290),
                                new Student("강자바", 3, 180)
                };

                // 학생 이름만 뽑아서 List<String>에 저장
                List<String> names = Stream.of(stuArr).map(Student::getName)
                                                .collect(Collectors.toList());
                System.out.println(names);

                // 스트림을 배열로 변환
                Student[] stuArr2 = Stream.of(stuArr).toArray(Student[]::new);

                for(Student s : stuArr2)
                        System.out.println(s);

                // 스트림을 Map<String, String>로 변환. 학생 이름 key
                Map<String, Student> stuMap = Stream.of(stuArr)
                                        .collect(Collectors.toMap(s->s.getName(), p->

                for(String name : stuMap.keySet())
                        System.out.println(name + "-" + stuMap.get(name));

                long count = Stream.of(stuArr).collect(counting());
                long totalScore = Stream.of(stuArr)
                                        .collect(summingInt(Student::getTotalScore));
                System.out.println("count=" + count);
                System.out.println("totalScore=" + totalScore);

                totalScore = Stream.of(stuArr)
                                .collect(reducing(0, Student::getTotalScore, Integer:
                System.out.println("totalScore=" + totalScore);

                Optional<Student> topStudent = Stream.of(stuArr)
                                        .collect(maxBy(Comparator.comparingInt(Studen
                System.out.println("topStudent=" + topStudent.get());
```

```
                IntSummaryStatistics stat = Stream.of(stuArr)
                                .collect(summarizingInt(Student::getTotalScore));
                System.out.println(stat);

                String stuNames = Stream.of(stuArr).map(Student::getName)
                                        .collect(joining(",", "[", "]"));
                System.out.println(stuNames);

        }

}
```

- 실행 결과

```
[이자바, 김자바, 안자바, 박자바, 소자바, 나자바, 강자바]
[이자바, 3, 300]
[김자바, 1, 200]
[안자바, 2, 100]
[박자바, 2, 150]
[소자바, 1, 200]
[나자바, 3, 290]
[강자바, 3, 180]
안자바-[안자바, 2, 100]
김자바-[김자바, 1, 200]
박자바-[박자바, 2, 150]
나자바-[나자바, 3, 290]
강자바-[강자바, 3, 180]
이자바-[이자바, 3, 300]
소자바-[소자바, 1, 200]
count=7
totalScore=1420
totalScore=1420
topStudent=[이자바, 3, 300]
IntSummaryStatistics{count=7, sum=1420, min=100, average=202.857143, max=300}
[이자바,김자바,안자바,박자바,소자바,나자바,강자바]
```

그룹화와 분할 - groupingBy(), partitioningBy()

- 그룹화는 스트림의 요소를 특정 기준으로 그룹화하는 것을 의미한다.
- 분할은 스트림의 요소를 두 가지, 지정된 조건에 일치하는 그룹과 일치하지 않는 그룹으로 분할을 의미한다.
- grouping()은 스트림의 요소를 Function으로 분류한다.
- partitioningBy()은 Predicate로 분류한다.
- 스트림을 두개의 그룹으로 나눠야 한다면 partitioningBy()로 분할하는 것이 더 빠르다.
- 그 외에는 groupingBy()를 쓰면 된다.
- 그룹화와 분할의 결과는 Map에 담겨 반환된다.

```
Collector groupingBy(Function classifier)
Collector groupingBy(Function classifier, Collector downstream)
```

```
Collector groupingBy(Function classifier, Supplier mapFactory, Collector
downstream)

Collector partitioningBy(Predicate predicate)
Collector partitioningBy(Predicate predicate, Collector downstream)
```

partitioningBy()에 의한 분류

- 기본분할

```
Map<Boolean, List<Student>> stuBySex = stuStream

.collect(Collectors.partitioningBy(Student::isMale)); // 학생을 성별로 분할

List<Student> maleStudent = stuBySex.get(true); // Map에서 남학생 목록을 얻는다.
List<Student> femaleStudent = stuBySex.get(false); // Map에서 여학생 목록을 얻는다.
```

- 기본분할 + 통계정보

```
Map<Boolean, Long> stuNumBySex = stuStream

.collect(Collectors.partitioningBy(Student::isMale, Collectors.counting()));

System.out.println("남학생 수 :" + stuNumBySex.get(true));
System.out.println("여학생 수 :" + stuNumBySex.get(false));
```

- 이중 분할

```
Map<Boolean, Map<Boolean, List<Student>>> failedStuBySex = stuStream
                .collect(
                        partitioningBy(Student::isMale,
                                partitioningBy(s -> s.getScore() < 150)
                        )
                );

List<Student> failedMaleStu = failedStuBySex.get(true).get(true);
List<Student> failedFemaleStu = failedStuBySex.get(false).get(true);
```

day19/collect/Student.java

```
package day19.collect;

public class Student {
        String name; // 이름
        boolean isMale; // 성별
        int hak; // 학년
        int ban; // 반
        int score; // 점수

        Student(String name, boolean isMale, int hak, int ban, int score) {
```

```
                this.name = name;
                this.isMale = isMale;
                this.hak = hak;
                this.ban = ban;
                this.score = score;
        }

        String getName() { return name; }
        boolean isMale() { return isMale; }
        int getHak() { return hak; }
        int getBan() { return ban; }
        int getScore() { return score; }

        public String toString() {
                return String.format("[%s, %s, %d학년 %d반, %3d점]", name, isMale?"남"
        }

        enum Level { HIGH, MID, LOW } // 성적을 상, 중, 하 세 단계로 분류
}
```

day19.collect/StreamEx7.java

```
package day19.collect;

import java.util.*;
import java.util.stream.*;
import static java.util.stream.Collectors.*;
import static java.util.Comparator.*;

public class StreamEx7 {
        public static void main(String[] args) {
                Student[] stuArr = {
                        new Student("나자바", true, 1, 1, 300),
                        new Student("김지미", false, 1, 1, 250),
                        new Student("김자바", true, 1, 1, 200),
                        new Student("이지미", false, 1, 2, 150),
                        new Student("남자바", true, 1, 2, 100),
                        new Student("안지미", false, 1, 2, 50),
                        new Student("황지미", false, 1, 3, 100),
                        new Student("강지미", false, 1, 3, 150),
                        new Student("이자바", true, 1, 3, 200),

                        new Student("나자바", true, 2, 1, 300),
                        new Student("김지미", false, 2, 1, 250),
                        new Student("김자바", true, 2, 1, 200),
                        new Student("이지미", false, 2, 2, 150),
                        new Student("남자바", true, 2, 2, 100),
                        new Student("안지미", false, 2, 2, 50),
                        new Student("황지미", false, 2, 3, 100),
                        new Student("강지미", false, 2, 3, 150),
                        new Student("이자바", true, 2, 3, 200),
                };

                System.out.print("1. 단순분할 (성별로 분할)%n");
```

```
                Map<Boolean, List<Student>> stuBySex = Stream.of(stuArr)
                                                        .collect(partitioningBy(Stude

                List<Student> maleStudent = stuBySex.get(true);
                List<Student> femaleStudent = stuBySex.get(false);

                for(Student s : maleStudent) System.out.println(s);
                for(Student s : femaleStudent) System.out.println(s);

                System.out.printf("%n2. 단순분할 + 통계(성별 학생수)%n");
                Map<Boolean, Long> stuNumBySex = Stream.of(stuArr)
                                                .collect(partitioningBy(Student::isMa

                System.out.println("남학생수 :" + stuNumBySex.get(true));
                System.out.println("여학생수 :" + stuNumBySex.get(false));

                System.out.printf("%n3. 단순분할 + 통계(성별 1등)%n");
                Map<Boolean, Optional<Student>> topScoreBySex = Stream.of(stuArr)
                                        .collect(partitioningBy(Student::isMale,
                                                        maxBy(comparingInt(Student::g

                System.out.println("남학생 1등 :" + topScoreBySex.get(true));
                System.out.println("여학생 1등 :" + topScoreBySex.get(false));

                Map<Boolean, Student> topScoreBySex2 = Stream.of(stuArr)
                                        .collect(partitioningBy(Student::isMale, coll

                System.out.println("남학생 1등 :" + topScoreBySex2.get(true));
                System.out.println("여학생 1등 :" + topScoreBySex2.get(false));

                System.out.printf("%n4. 다중분할(성별 불합격자, 100점 이하)%n");

                Map<Boolean, Map<Boolean, List<Student>>> failedStuBySex = Stream.of(
                List<Student> failedMaleStu = failedStuBySex.get(true).get(true);
                List<Student> failedFemaleStu = failedStuBySex.get(false).get(true);

                for(Student s : failedMaleStu) System.out.
                println(s);
                for(Student s : failedFemaleStu) System.out.println(s);
        }
}
```

- 실행결과

```
1. 단순분할 (성별로 분할)%n[나자바, 남, 1학년 1반, 300점]
[김자바, 남, 1학년 1반, 200점]
[남자바, 남, 1학년 2반, 100점]
[이자바, 남, 1학년 3반, 200점]
[나자바, 남, 2학년 1반, 300점]
[김자바, 남, 2학년 1반, 200점]
[남자바, 남, 2학년 2반, 100점]
[이자바, 남, 2학년 3반, 200점]
[김지미, 여, 1학년 1반, 250점]
[이지미, 여, 1학년 2반, 150점]
```

```
[안지미, 여, 1학년 2반,  50점]
[황지미, 여, 1학년 3반, 100점]
[강지미, 여, 1학년 3반, 150점]
[김지미, 여, 2학년 1반, 250점]
[이지미, 여, 2학년 2반, 150점]
[안지미, 여, 2학년 2반,  50점]
[황지미, 여, 2학년 3반, 100점]
[강지미, 여, 2학년 3반, 150점]

2. 단순분할 + 통계(성별 학생수)
남학생수 :8
여학생수 :10

3. 단순분할 + 통계(성별 1등)
남학생 1등 :Optional[[나자바, 남, 1학년 1반, 300점]]
여학생 1등 :Optional[[김지미, 여, 1학년 1반, 250점]]
남학생 1등 :[나자바, 남, 1학년 1반, 300점]
여학생 1등 :[김지미, 여, 1학년 1반, 250점]

4. 다중분할(성별 불합격자, 100점 이하)
[남자바, 남, 1학년 2반, 100점]
[남자바, 남, 2학년 2반, 100점]
[안지미, 여, 1학년 2반,  50점]
[황지미, 여, 1학년 3반, 100점]
[안지미, 여, 2학년 2반,  50점]
[황지미, 여, 2학년 3반, 100점]
```

groupingBy()에 의한 분류

- groupingBy()로 그룹화 하면 기본적으로 List에 담는다.
- toList()대신 toSet()이나 toCollection(HashSet::new)을 사용할 수 도 있다.
- Map의 제네릭 타입도 적절하게 변경해야 한다.
- 기본 그룹화

```
Map<Integer, List<Student>> stuByBan = stuStream
                .collect(Collectors.groupingBy(Student::getBan)); // toList()가 생
략됨
```

- 기본 그룹화 + 통계정보

```
Map<Student.Level, Long> stuByLevel = stuStream
                .collect(groupingBy(s -> {
                        if (s.getScore() >= 200) return Student.Level.HIGH;
                        else if (s.getScore() >= 100) return Student.Level.MID;
                        else return Student.Level.LOW;
                }, counting()));   // [MID] - 0명, [HIGH] - 8명, [LOW] - 2명
```

- 다중 그룹화

```
Map<Integer, Map<Integer, List<Student>>> stuByHakAndBan = stuStream
                .collect(groupingBy(Student::getHak,
groupingBy(Student::getBan)));
```

day19/collect/StreamEx8.java

```
package day19.collect;

import java.util.*;
import java.util.stream.*;
import static java.util.stream.Collectors.*;
import static java.util.Comparator.*;

public class StreamEx8 {

        public static void main(String[] args) {
                Student[] stuArr = {
                                new Student("나자바", true, 1, 1, 300),
                                new Student("김지미", false, 1, 1, 250),
                                new Student("김자바", true, 1, 1, 200),
                                new Student("이지미", false, 1, 2, 150),
                                new Student("남자바", true, 1, 2, 100),
                                new Student("안지미", false, 1, 2, 50),
                                new Student("황지미", false, 1, 3, 100),
                                new Student("강지미", false, 1, 3, 150),
                                new Student("이자바", true, 1, 3, 200),

                                new Student("나자바", true, 2, 1, 300),
                                new Student("김지미", false, 2, 1, 250),
                                new Student("김자바", true, 2, 1, 200),
                                new Student("이지미", false, 2, 2, 150),
                                new Student("남자바", true, 2, 2, 100),
                                new Student("안지미", false, 2, 2, 50),
                                new Student("황지미", false, 2, 3, 100),
                                new Student("강지미", false, 2, 3, 150),
                                new Student("이자바", true, 2, 3, 200),
                        };
                System.out.printf("1. 단순그룹화(반별로 그룹화)%n");
                Map<Integer, List<Student>> stuByBan = Stream.of(stuArr)
                                        .collect(groupingBy(Student::getBan));

                for (List<Student> ban : stuByBan.values()) {
                        for (Student s : ban) {
                                System.out.println(s);
                        }
                }

                System.out.printf("%n2. 단순그룹화(성별로 그룹화)%n");
                Map<Student.Level, List<Student>> stuByLevel = Stream.of(stuArr)
                                        .collect(groupingBy(s -> {
                                                        if (s.getScore() >= 200) retu
                                                        else if (s.getScore() >= 100)
                                                        else return Student.Level.LOW
                                        }));
```

```
TreeSet<Student.Level> keySet = new TreeSet<>(stuByLevel.keySet());

for(Student.Level key : keySet) {
        System.out.println("[" + key + "]");

        for (Student s : stuByLevel.get(key))
                System.out.println(s);
        System.out.println();
}

System.out.printf("%n3. 단순그룹화 + 통계(성적별 학생수)%n");
Map<Student.Level, Long> stuCntByLevel = Stream.of(stuArr)
                        .collect(groupingBy(s -> {
                                        if (s.getScore() >= 200) retu
                                        else if(s.getScore() >= 100)
                                        else return Student.Level.LOW
                        }, counting()));

for (Student.Level key : stuCntByLevel.keySet())
        System.out.printf("[%s] - %d명, ", key, stuCntByLevel.get(key
System.out.println();

System.out.printf("%n4. 다중그룹화(학년별, 반별)");
Map<Integer, Map<Integer, List<Student>>> stuByHakAndBan = Stream.of(
                                .collect(groupingBy(Student::getHak,

for (Map<Integer, List<Student>> hak : stuByHakAndBan.values()) {
        for(List<Student> ban : hak.values()) {
                System.out.println();
                for(Student s : ban)
                        System.out.println(s);
        }
}

System.out.printf("%n5. 다중그룹화 + 통계(학년별, 반별 1등)%n");
Map<Integer, Map<Integer, Student>> topStuByHakAndBan = Stream.of(stu
                .collect(groupingBy(Student::getHak,
                                groupingBy(Student::getBan,
                                                collectingAndThen(
                                                        maxBy(compari
                                                )
                                )
        ));

for(Map<Integer, Student> ban : topStuByHakAndBan.values())
        for(Student s : ban.values())
                System.out.println(s);

System.out.printf("%n6. 다중그룹화 + 통계(학년별, 반별 성적그룹)%n");
Map<String, Set<Student.Level>> stuByScoreGroup = Stream.of(stuArr)
                        .collect(groupingBy(s->s.getHak() + "-" + s.g
                                        mapping(s -> {
                                                if (s.getScore() >= 2
                                                else if (s.getScore()
                                                else return Student.L
```

```
                                                }, toSet())
                                ));
                Set<String> keySet2 = stuByScoreGroup.keySet();

                for(String key : keySet2) {
                        System.out.println("[" + key + "]" + stuByScoreGroup.get(key)
                }
        }
}
```

- 실행결과

```
1. 단순그룹화(반별로 그룹화)
[나자바, 남, 1학년 1반, 300점]
[김지미, 여, 1학년 1반, 250점]
[김자바, 남, 1학년 1반, 200점]
[나자바, 남, 2학년 1반, 300점]
[김지미, 여, 2학년 1반, 250점]
[김자바, 남, 2학년 1반, 200점]
[이지미, 여, 1학년 2반, 150점]
[남자바, 남, 1학년 2반, 100점]
[안지미, 여, 1학년 2반,  50점]
[이지미, 여, 2학년 2반, 150점]
[남자바, 남, 2학년 2반, 100점]
[안지미, 여, 2학년 2반,  50점]
[황지미, 여, 1학년 3반, 100점]
[강지미, 여, 1학년 3반, 150점]
[이자바, 남, 1학년 3반, 200점]
[황지미, 여, 2학년 3반, 100점]
[강지미, 여, 2학년 3반, 150점]
[이자바, 남, 2학년 3반, 200점]

2. 단순그룹화(성별로 그룹화)
[HIGH]
[나자바, 남, 1학년 1반, 300점]
[김지미, 여, 1학년 1반, 250점]
[김자바, 남, 1학년 1반, 200점]
[이자바, 남, 1학년 3반, 200점]
[나자바, 남, 2학년 1반, 300점]
[김지미, 여, 2학년 1반, 250점]
[김자바, 남, 2학년 1반, 200점]
[이자바, 남, 2학년 3반, 200점]

[MID]
[이지미, 여, 1학년 2반, 150점]
[남자바, 남, 1학년 2반, 100점]
[황지미, 여, 1학년 3반, 100점]
[강지미, 여, 1학년 3반, 150점]
[이지미, 여, 2학년 2반, 150점]
[남자바, 남, 2학년 2반, 100점]
[황지미, 여, 2학년 3반, 100점]
[강지미, 여, 2학년 3반, 150점]

[LOW]
```

```
[안지미, 여, 1학년 2반,  50점]
[안지미, 여, 2학년 2반,  50점]


3. 단순그룹화 + 통계(성적별 학생수)
[LOW] - 2명, [HIGH] - 8명, [MID] - 8명,

4. 다중그룹화(학년별, 반별)
[나자바, 남, 1학년 1반, 300점]
[김지미, 여, 1학년 1반, 250점]
[김자바, 남, 1학년 1반, 200점]

[이지미, 여, 1학년 2반, 150점]
[남자바, 남, 1학년 2반, 100점]
[안지미, 여, 1학년 2반,  50점]

[황지미, 여, 1학년 3반, 100점]
[강지미, 여, 1학년 3반, 150점]
[이자바, 남, 1학년 3반, 200점]

[나자바, 남, 2학년 1반, 300점]
[김지미, 여, 2학년 1반, 250점]
[김자바, 남, 2학년 1반, 200점]

[이지미, 여, 2학년 2반, 150점]
[남자바, 남, 2학년 2반, 100점]
[안지미, 여, 2학년 2반,  50점]

[황지미, 여, 2학년 3반, 100점]
[강지미, 여, 2학년 3반, 150점]
[이자바, 남, 2학년 3반, 200점]

5. 다중그룹화 + 통계(학년별, 반별 1등)
[나자바, 남, 1학년 1반, 300점]
[이지미, 여, 1학년 2반, 150점]
[이자바, 남, 1학년 3반, 200점]
[나자바, 남, 2학년 1반, 300점]
[이지미, 여, 2학년 2반, 150점]
[이자바, 남, 2학년 3반, 200점]

6. 다중그룹화 + 통계(학년별, 반별 성적그룹)
[1-1][HIGH]
[2-1][HIGH]
[1-2][LOW, MID]
[2-2][LOW, MID]
[1-3][HIGH, MID]
[2-3][HIGH, MID]
```

스트림의 변환

1. 스트림 -> 기본형 스트림

from	to	변환 메서드
Stream<T>	IntStream LongStream DoubleStream	mapToInt(ToIntFunction<T> mapper) mapToLong(ToLongFunction<T> mapper) mapToDouble(ToDoubleFunction<T> mapper)

2. 기본형 스트림 -> 스트림

from	to	변환 메서드
IntStream LongStream DoubleStream	Stream<Integer> Stream<Long> Stream<Double> Stream<U>	boxed() boxed() boxed() mapToObj(DoubleFunction<U> mapper)

3. 기본형 스트림 -> 기본형 스트림

from	to	변환 메서드
Stream<T>	IntStream LongStream DoubleStream	LongStream DoubleStream

4. 스트림 -> 부분 스트림

from	to	변환 메서드
Stream<T> IntStream	Stream<T> IntStream	skip(long n) limit(long maxSize)

5. 두 개의 스트림 -> 스트림

from	to	변환 메서드
Stream<T>, Stream<T>	Stream<T>	concat(Stream<T> a, Stream<T>b)
IntStream, IntStream	IntStream	concat(IntStream a, IntStream b)
LongStream, LongStream	LongStream	concat(LongStream a, LongStream b)
DoubleStream, DoubleStream	DoubleStream	concat(DoubleStream a, DoubleStream b)

6. 스트림의 스트림 -> 스트림

from	to	변환 메서드
Stream<\Stream<T>>	Stream<T>	flatMap(Function mapper)
Stream<IntStream>	IntStream	flatMapToInt(Function mapper)
Stream<LongStream>	LongStream	flatMapToLong(Function mapper)

from	to	변환 메서드
Stream<DoubleStream>	DoubleStream	flatMapToDouble(Function mapper)

7. 스트림 -> 병렬 스트림

from	to	변환 메서드

Stream<T>
IntStream
LongStream
DoubleStream
	Stream<T>
	IntStream
	LongStream
	DoubleStream
		parallel() // 스트림 -> 병렬 스트림
		sequential() // 병렬 스트림 -> 스트림

8. 스트림 -> 컬렉션

from	to	변환 메서드
Stream<T> IntStream LongStream DoubleStream	Collection<T> List<T> Set<T>	collect(Collectors.toCollection(Supplier supplier)) collect(Collectors.toList()) Collectors.toSet())

9. 컬렉션 -> 스트림

from	to	변환 메서드
Collection<T>, List<T>, Set<T>	Stream<T>	stream()

10. 스트림 -> Map

from	to	변환 메서드
Stream<T> IntStream LongStream DoubleStream	Map<K,V>	collect(Collectors.toMap(Function Key, Function value)) collect(Collectors.toMap(Function k, Function v, BinaryOperator merge))

11. 스트림 -> 배열

from	to	변환 메서드
Stream<T>	Object[] T[]	toArray() toArray(intFunction<A[]> generator)
IntStream LongStream DoubleStream	int[] long[] double[]	toArray()