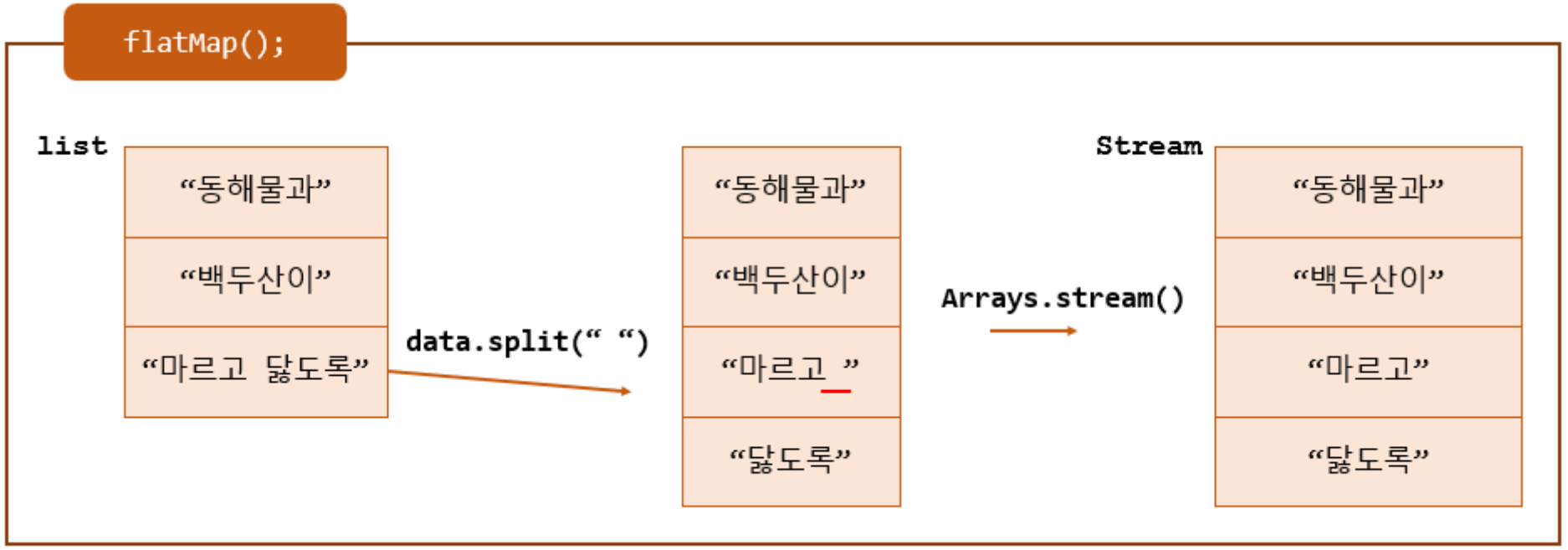
5/10 스터디노트

- P450 스트림 가공하기 2 : Mapping 매핑
  - ✓ 스트림의 데이터를 매개변수로 받아서 다른 타입으로 리턴함
  - ✓ `flatmap();` 객체를 입력받아서 Stream 으로 리턴

```
// 문자열을 공백으로 분리해서 매핑
List<String> list1 = Arrays.asList("동해물과", "백두산이",
                                   "마르고 닳도록");
list1.stream().flatMap(data -> Arrays.stream(data.split( regex: " ")))
              .forEach(word -> System.out.println(word));
```



✓ flatMap 선생님 예제

```
public static void main(String[] args) {
    String[][] names = {  {"John", "Mary"}, {"Alice","Bob"}, {"Tom","Jerry"}  };

    Stream<String[]> stream = Arrays.stream(names);
    Stream<String> flatStream = stream.flatMap(s -> Arrays.stream(s));

    flatStream.forEach(n -> System.out.println(n));
}
```

```
John
Mary
Alice
Bob
Tom
Jerry
```

- P453

✓ `mapToInt();` 객체를 입력받아서 int 타입 Stream 으로 리턴

```
List<String> list = Arrays.asList("동해물과", "백두산이", "마르고 닳도록");

System.out.println("함수적 인터페이스 방식");
list.stream().mapToInt(s -> s.length()).forEach(len-> System.out.println(len));

System.out.println();

System.out.println("메서드 참조 방식");
list.stream().mapToInt(String::length).forEach(len-> System.out.println(len));
```

람다식 `s -> s.length()` 를

[클래스명]::[클래스의 메서드명] 의 형태로도 사용가능

```
함수적 인터페이스 방식
4
4
7

메서드 참조 방식
4
4
7
```

- P454
  - ✓ `asDoubleStream();` IntStream 이나 LongStream 을 DoubleStream 으로 형변환
  - ✓ `boxed();` IntStream, LongStream, DoubleStream
    -> Stream<Integer> / Stream<Long> / Stream<Double> 형변환

기본타입 -> Wrapper 클래스 타입 : Boxing

Wrapper 클래스 타입 -> 기본타입 : Unboxing

```
int intArr[] = {10, 20, 30, 40, 50, 60};

Arrays.stream(intArr).asDoubleStream().forEach(d -> System.out.println(d));
    intArr배열을 stream으로      IntStream을 Stream<Double> 로        출력
System.out.println();

Arrays.stream(intArr).boxed().forEach(i -> System.out.println(i.getClass()));
                                                                  클래스 정보를 출력
```

```
10.0
20.0
30.0
40.0
50.0
60.0

class java.lang.Integer
class java.lang.Integer
class java.lang.Integer
class java.lang.Integer
class java.lang.Integer
class java.lang.Integer
```

● P455 스트림 가공하기 3 : Sorting 정렬

✓ 정렬하기 위해서는 Comparable 인터페이스를 구현한 클래스의 객체여야함. (String 클래스는 Comparable을 구현했으므로 정렬가능.)
-> 내가 만든 클래스의 객체를 정렬하고 싶으면 클래스에서 Comparable 인터페이스를 구현하면 됨

```
List<String> list = Arrays.asList("홍길동", "김유신", "이순신", "유관순");

                                    단축키 soutc
list.stream().sorted().forEach(System.out::println);

list.stream().sorted(Comparator.reverseOrder()).forEach(System.out::println);
```

김유신
유관순
이순신
홍길동

홍길동
이순신
유관순
김유신

역순정렬

```
// 정렬과 같은 작업을 하기 위해서는 비교대상의 선후관계를 정의해야 한다.
// 이 작업을 반드시 강제하기 위해서 Comparable이라는 인터페이스를 구현해야한다.
// 그러면 추후에 정렬 관련 메소드에서 비교를 어떻게 하는지 Comparable의 추상메소드
// compareTo 메소드를 참조하게 된다.
```

```
public int compareTo(Shape s) {
    // 자신과 비교대상을 기준으로 같으면 0
    // 오름차순일 경우에 자신이 비교대상보다 작으면 음수 크면 양수로 반환
    // 내림차순일 경우에 자신이 비교대상보다 작으면 양수 크면 음수로 반환
    // 위 기준으로 사용자 정의 클래스 일 경우에 비교대상과 연산관련 로직을 정의한다.
    return (int)(this.area() - s.area() );
    // 따라서 기본 정렬을 내림차순으로 하고 싶다면 기본 오름차순 정렬 결과에 -1을 곱하면 된다.
    return (int)( (this.area() - s.area()) );
}
```

```
sorted((a,b) -> b.compareTo(a) - a.compareTo(b) )
```

- P456 – P458 교재 : 내가 만든 클래스의 객체를 정렬하기 (Comparable 인터페이스 구현 필요)

```
abstract class Shape implements Comparable<Shape> {
//필드
    int x,y;
//생성자
    Shape(){this( x: 0, y: 0);}
    Shape(int x, int y) { this.x = x;      this.y = y;  }
//추상메서드
    abstract double area();
//Comparable 인터페이스의 메서드 compareTo 재정의
    @Override
    public int compareTo(Shape s){
        return (int)(this.area() - s.area());
    }
}
```

compareTo 메서드 정의 : 객체(값) - 입력받은 객체(값) 연산결과 리턴

ex) `b.compareTo(a) = b – a`

```
public class Rectangle extends Shape {
//필드
    int w,h;
//생성자
    Rectangle(int w, int h){ this.w = w;  this.h = h; }
//메서드 오버라이딩
    @Override
    double area(){  return (w*h); }  //면적 구하기
    @Override
    public String toString(){  return "넓이: "+this.area();  }  //출력형태
}
```

```java
public class StreamOrder2 {
    public static void main(String[] args) {

        Shape s1 = new Rectangle( w: 10, h: 3);
        Shape s2 = new Rectangle( w: 10, h: 5);
        Shape s3 = new Rectangle( w: 10, h: 7);

        List<Shape> list = Arrays.asList(s1, s2, s3);

        list.stream().sorted().forEach(System.out::println);

        list.stream().sorted(Comparator.reverseOrder()).forEach(System.out::println);

        list.stream().sorted((a,b) -> b.compareTo(a) - a.compareTo(b)).forEach(System.out::println);

        list.stream().sorted((a,b) -> a.compareTo(b) * -1 ).forEach(System.out::println);

        list.stream().sorted((a,b) -> b.compareTo(a) * -1 ).forEach(System.out::println);
```

```
넓이: 30.0
넓이: 50.0
넓이: 70.0
```

```
넓이: 70.0
넓이: 50.0
넓이: 30.0
```

compareTo 메서드 정의 : 객체(값) - 입력받은 객체(값) 연산결과 리턴 **b.compareTo(a) = b - a**

sorted((a,b) -> b.compareTo(a) - a.compareTo(b))

compareTo 메서드 정의 적용하면 ➡ sorted((a,b) -> (b - a) - (a - b))

a 가 s1, b가 s2 라고 생각하면 ➡ sorted((s1,s2)-> (s2 - s1) - (s1 - s2))

s1 = 30, s2 = 50 이니까 ➡ sorted((s1,s2) -> (20) - (-20))

연산결과 ➡ **sorted**((s1,s2) -> 40) ➡ 결과가 **양수**이므로 s1 과 s2 **객체의 순서를 바꾸지 않고** s1, s2 의 순서 그대로 출력함

➡만약 a 자리에 s2, b 자리에 s1 을 넣어서 연산결과가 **음수**라면 **a,b 의 순서를 b,a 로 바꾸어** s1이 s2 보다 먼저 출력되도록 정렬됨

- P460 스트림 반복자 peek() : 중간단계에서 전체를 반복하면서 추가작업을 하기위해 사용함.
  - ✓ 중간 처리 작업을 하는 메서드이므로 코드 끝에 최종 처리 메서드를 호출해야 동작함 (최종 처리 : forEach(), sum() 등)
  - ✓ 선생님 예제

```java
public class P460_StreamPeek2_hk {
    public static void main(String[] args) {
        List<Integer> numbers = Arrays.asList(1,2,3,4,5);
        //peek 메서드는 주로 스트림의 중간 작업이 진행시 디버그 용도로 활용

        List<Integer> result = numbers.stream()
                .peek(num -> System.out.println("Processing number: " + num))
                .filter(num -> num % 2 == 0)
                .peek(num -> System.out.println("Even number: "+num))
                .collect(Collectors.toList()); → (최종 처리 메서드)
    }
}
```

```
Processing number: 1
Processing number: 2
Even number: 2
Processing number: 3
Processing number: 4
Even number: 4
Processing number: 5
```

list 요소 1부터 5까지 peek(num -> System.out.println("Processing number: " + num)) 을 반복함

그 사이에 filter 메서드로 2로 나눈 나머지가 0인 경우 (=짝수인 경우)

peek(num -> System.out.println("Even number: " + num)) 반복

✓ P460 StreamPeek 교재 예제

```
Shape s1 = new Rectangle( w: 10,  h: 3);
Shape s2 = new Circle( r: 10);
Shape s3 = new Rectangle( w: 20,  h: 2);
Shape s4 = new Circle( r: 11);

List<Shape> list = Arrays.asList(s1, s2, s3, s4);

list.parallelStream().mapToDouble(a -> a.area()).peek(a-> System.out.println(a)).sum();
```

```
40.0
380.132711084365
314.1592653589793
30.0
```

(최종 처리 메서드)

- P462 요소 조건 검사 : 스트림 내부의 요소가 어떤 조건을 만족하는지 여부를 검사

메서드	설명
allMatch(Predicate p)	모두 p 조건 만족 : true / 하나라도 불만족 : false
anyMatch(Predicate p)	하나라도 p 조건 만족 : true / 모두 불만족 : false
noneMatch(Predicate p)	모두 p 조건 불만족 : true / 하나라도 만족 : false

```
public static void main(String[] args) {
    Shape s1 = new Rectangle( w: 10,  h: 3);
    Shape s2 = new Circle( r: 10);
    Shape s3 = new Rectangle( w: 20,  h: 2);
    Shape s4 = new Circle( r: 11);

    List<Shape> list = Arrays.asList(s1, s2, s3, s4);

    boolean result = list.stream().allMatch(a -> (a instanceof Shape));
    System.out.println("list 스트림의 모든 요소는 Shape 의 객체이다 : "+result);

    boolean result2 = list.stream().anyMatch(a -> (a instanceof Rectangle));
    System.out.println("list 스트림의 요소 중 1개 이상은 Rectangle 의 객체이다 : "+result2);

    boolean result3 = list.stream().noneMatch(a -> (a instanceof Circle));
    System.out.println("list 스트림의 요소 중 하나도 Circle 의 객체가 아니다 : "+result3);
}
```

```
list 스트림의 모든 요소는 Shape 의 객체이다 : true
list 스트림의 요소 중 1개 이상은 Rectangle 의 객체이다 : true
list 스트림의 요소 중 하나도 Circle 의 객체가 아니다 : false
```

- P463 집계 메서드

✓ Stream 은 요소들의 최소 최대 합계 평균값, 개수 등을 구할 수 있는 메서드를 제공함

리턴타입	메서드	설명
long	count()	요소들의 갯수
int	sum()	IntStream 요소들의 합
long	sum()	LongStream 요소들의 합
double	sum()	DoubleStream 요소들의 합
OptionalXXX	findFirst()	첫번째 요소
Optionalxxx	max()	기본 정렬에서 최대값 요소
Optional<T>	max(Comparator<T>)	설정한 정렬에서 최대값 요소
OptionalXXX	min()	기본 정렬에서 최소값 요소
Optional<T>	min(Comparator<T>)	설정한 정렬에서 최소값 요소
OptionalDouble	average()	요소들의 평균

count나 sum 메서드는

요소가 없으면 결과값이 0

findFirst / max / min / average 는

요소가 없으면 리턴값도 없음..

그래도 예외가 발생하지 않고 안전하게 처리하기 위해 사용하는 타입 = Optional

종류: `Optional` / `OptionalInt` / `OptionalLong` / `OptionalDouble`

[Stream 사용하여 집계하기]

개수
```
long count = Arrays.stream(arr).count();
```

합
```
int sum = Arrays.stream(arr).sum();
```

최대값
```
OptionalInt max = Arrays.stream(arr).max();
```

평균값
```
OptionalDouble avg = Arrays.stream(arr).average();
```

첫번째값
```
OptionalInt first = Arrays.stream(arr).findFirst();
```

[Stream 사용하지 않고 집계하기]

개수
```
long count = arr.length;
```

첫번째값
```
int first = arr[0];
```

합
```
int sum = 0;
for(int i=0; i<arr.length; i++) {
    sum += arr[i];
}
```

최대값
```
int max = arr[0];
for(int i=1; i<arr.length; i++) {
    if(arr[i] > max) {
        max = arr[i];
    }
}
```

평균
```
double avg = sum / (double)count;
```

- P463 집계메서드 사용 예제

```java
int[] arr = new int[100];
for (int i = 0; i < 100; i++) {
    arr[i] = i+1;
}

long count = Arrays.stream(arr).count();
System.out.println("count() : " +count);
int sum = Arrays.stream(arr).sum();
System.out.println("sum() : " +sum);

OptionalInt first = Arrays.stream(arr).findFirst();
System.out.println("findFirst() : "+ first.getAsInt());

OptionalInt max = Arrays.stream(arr).max();
System.out.println("max() : "+max.getAsInt());

OptionalInt min = Arrays.stream(arr).min();
System.out.println("min(): " + min.getAsInt());

OptionalDouble avg = Arrays.stream(arr).average();
System.out.println("average(): "+avg.getAsDouble());
```

arr = { 1, 2, 3, ... , 98, 99, 100 }

```
count() : 100
sum() : 5050
findFirst() : 1
max() : 100
min(): 1
average(): 50.5
```

getAsXXX()

메서드 사용 안 한 경우 출력 결과

```java
OptionalInt first = Arrays.stream(arr).findFirst();
System.out.println("findFirst() : "+ first);

OptionalInt max = Arrays.stream(arr).max();
System.out.println("max() : "+max);

OptionalInt min = Arrays.stream(arr).min();
System.out.println("min(): " + min);

OptionalDouble avg = Arrays.stream(arr).average();
System.out.println("average(): "+avg);
```

```
count() : 100
sum() : 5050
findFirst() : OptionalInt[1]
max() : OptionalInt[100]
min(): OptionalInt[1]
average(): OptionalDouble[50.5]
```

- P465 요소가 없는 예제 – 오류 발생

```java
List<Integer> list = new ArrayList<Integer>();  //요소가 없는 빈 객체

long count = list.stream().count();
System.out.println("count() : " + count);

int sum = list.stream().mapToInt(Integer::intValue).sum();
System.out.println("sum() : " + sum);

OptionalDouble avg = list.stream().mapToInt(Integer::intValue).average();
System.out.println("average() : " + avg.getAsDouble());

OptionalInt max = list.stream().mapToInt(Integer::intValue).max();
System.out.println("max() : " + max.getAsInt());

OptionalInt min = list.stream().mapToInt(Integer::intValue).min();
System.out.println("min() : " + min.getAsInt());

OptionalInt first = list.stream().mapToInt(Integer::intValue).findFirst();
System.out.println("findFirst() : " + first.getAsInt());
```

```
count() : 0
sum() : 0
Exception in thread "main" java.util.NoSuchElementException Create breakpoint : No value present
    at java.base/java.util.OptionalDouble.getAsDouble(OptionalDouble.java:130)
    at chapter16.P464_StreamOptionalNoElem.main(P464_StreamOptionalNoElem.java:17)
```

- P466 요소가 없는 예제 2 – 요소값이 없는 경우 기본값을 출력하도록 설정

```java
// 요소가 없는 빈 ArrayList 객체 생성
List<Integer> list = new ArrayList<Integer>();

long count = list.stream().count();
System.out.println("요소들의 갯수 : " + count);

int sum = list.stream().mapToInt(Integer::intValue).sum();
System.out.println("요소들의 합 : " + sum);

OptionalDouble avg = list.stream().mapToInt(Integer::intValue).average();

// 요소가 존재하는 경우에만 평균 출력
if(avg.isPresent()) {
    System.out.println("요소들의 평균: " + avg.getAsDouble());
}

// 요소값이 없는 경우 기본값 설정
int max = list.stream().mapToInt(Integer::intValue).max().orElse( other: 0);
System.out.println("요소들 중 최대 값 : " + max);

int min = list.stream().mapToInt(Integer::intValue).min().orElse( other: -1);
System.out.println("요소들 중 최소값 : " + min);

// 요소가 존재하면 실행
list.stream().mapToInt(Integer::intValue).findFirst().ifPresent(a -> System.out.println("요소 중 첫번째 값 : " + a));
```

```
요소들의 갯수 : 0
요소들의 합 : 0
요소들 중 최대 값 : 0
요소들 중 최소값 : -1
```

- P467 사용자 집계 메서드 : reduce() 사용자가 원하는 방식으로 집계할 수 있음

```
Shape s1 = new Rectangle( w: 10,  h: 3);
Shape s2 = new Circle( r: 10);
Shape s3 = new Rectangle( w: 20,  h: 2);
Shape s4 = new Circle( r: 11);

List<Shape> list = Arrays.asList(s1, s2, s3, s4);

double areaSum = list.stream().mapToDouble(Shape::area).sum();
System.out.println("sum() 을 이용한 면적 합계 : " + areaSum);

//reduce({BinaryOperator<T>})
//a: 이전 요소,  b: 현재 요소

areaSum = list.stream().mapToDouble(Shape::area).reduce( (a,b) -> a+b ).getAsDouble();
System.out.println("reduce(Operator) 를 이용한 면적 합계 : "+areaSum);

areaSum = list.stream().mapToDouble(Shape::area).reduce( identity: 0,(a,b)->a+b );
System.out.println("reduce(0,Operator) 를 이용한 면적 합계 : "+areaSum);
```

s1 + s2 + s3 + s4

0 + s1 + s2 + s3 + s4

- **P468 `collect()` 메서드**
  - ✓ 스트림에서 요소들을 필터링하거나 매핑 (중간작업) 후에 **새로운 객체로 생성**하는 메서드 (Collectors.toList / Collectors.toSet)

```
rectList
넓이: 30.0
넓이: 40.0
rectSet
넓이: 40.0
넓이: 30.0
```

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

    Shape s1 = new Rectangle( w: 10,  h: 3);
    Shape s2 = new Circle( r: 10);
    Shape s3 = new Rectangle( w: 20,  h: 2);
    Shape s4 = new Circle( r: 11);

    List<Shape> list = Arrays.asList(s1, s2, s3, s4);


// 요소가 Rectangle 객체인 경우 collect 메서드로 List 객체인 rectList 로 변환하기
    System.out.println("rectList");
    List<Shape> rectList = list.stream().filter(s -> s instanceof Rectangle).collect(Collectors.toList());
                           //생성    //Rectangle 객체만 필터링            //필터링 된 요소들로  새로운 List 객체 생성
    rectList.stream().forEach(System.out::println);


// 요소가 Rectangle 객체인 경우 collect 메서드로 Set 객체인 rectSet 으로 변환
    System.out.println("rectSet");
    Set<Shape> rectSet = list.stream().filter(s -> s instanceof Rectangle).collect(Collectors.toSet());
                         //생성    //Rectangle 객체만 필터링            //필터링 된 요소들로  새로운 Set 객체 생성
    rectSet.stream().forEach(System.out::println);
}
```

- P469 **groupingBy()** 메서드
  - ✓ 특정 기준으로 종류별로 묶기

```
Shape s1 = new Rectangle( w: 10,  h: 3);
Shape s2 = new Circle( r: 10);
Shape s3 = new Rectangle( w: 20,  h: 2);
Shape s4 = new Circle( r: 11);

List<Shape> list = Arrays.asList(s1, s2, s3, s4);

try {

    Map<Object, List<Shape>> map = list.stream().collect(Collectors.groupingBy(f -> f.getClass()));

    System.out.println("사각형");      클래스명이 Rectangle 인 객체로 stream 생성해서 출력
    map.get(Class.forName( className: "chapter16.Rectangle")).stream().forEach(System.out::println);

    System.out.println("원");          클래스명이 Circle 인 객체로 stream 생성해서 출력
    map.get(Class.forName( className: "chapter16.Circle")).stream().forEach(System.out::println);

} catch (ClassNotFoundException e) {
    System.out.println(e.getMessage());
}
```

클래스 같은 것 끼리 그룹화

```
사각형
넓이: 30.0
넓이: 40.0
원
넓이: 314.1592653589793
넓이: 380.13271084365
```

- P470 스트림 병렬처리 : **parallelStream()** 메서드 이용

✓ parallelStream() 로 스트림 생성하면 그냥 stream() 으로 생성한 것 보다
데이터 처리할 시에 내부적으로 순차적이 아니라 개별 쓰레드로 처리해서 처리속도가 빨라짐 ~!

```
List<Integer> numbers = new ArrayList<>();
// 1부터 100,000,000까지의 정수 리스트 생성
for (int i = 0; i < 100000000; i++) {
    numbers.add(i);
}
long startTime = System.currentTimeMillis();

//Stream을 사용한 경우
int sum = numbers.stream().mapToInt(Integer::intValue).sum();

long endTime = System.currentTimeMillis();
System.out.println("Stream sum: " + sum);
// 1s: 1000ms
System.out.println("Stream elapsed time: " + (endTime - startTime) + "ms");

startTime = System.currentTimeMillis();

// parallelStream을 사용한 경우
sum = numbers.parallelStream().mapToInt(Integer::intValue).sum();

endTime = System.currentTimeMillis();
System.out.println("ParallelStream sum: " + sum);
System.out.println("ParallelStream elapsed time:" + (endTime-startTime) + "ms");
```

```
Stream sum: 887459712
Stream elapsed time: 155ms
ParallelStream sum: 887459712
ParallelStream elapsed time:110ms
```

- student management 프로그램 스트림으로 작성해보기
  - ✓ 메뉴 수정하기

```java
//메뉴 출력 및 기능 실행
private static void showMenu(){
    Scanner scanner = new Scanner(System.in);

    while(true){
        System.out.println("\n================================");
        System.out.println("1. 전체 학생 정보");
        System.out.println("2. 최고 평균 점수 학생 검색");
        System.out.println("3. 최저 평균 점수 학생 검색");
        System.out.println("4. 학생 검색");
        System.out.println("5. 학생 추가");
        System.out.println("6. 종료");
        System.out.println("\n================================");
        System.out.print("원하는 메뉴를 선택하세요: ");

        int menu = scanner.nextInt();
        scanner.nextLine();

        switch (menu){
            case 1: printAllStudents();             break;
            case 2: searchHighestAverageStudent();  break;
            case 3: searchLowestAverageStudent();   break;
            case 4: searchStudent();                break;
            case 5: addStudent();                   break;
            case 6:
                System.out.println("프로그램을 종료합니다.");
                scanner.close();
                System.exit( status: 0);
            default:
                System.out.println("잘못된 메뉴를 선택하셨습니다. 다시 선택해주세요.");
        }
    }
}
```

```java
//메뉴 출력 및 기능 실행
private static void showMenu(){
    Scanner scanner = new Scanner(System.in);

    while(true){
        System.out.println("\n================================");
        System.out.println("1. 전체 학생 정보");
        System.out.println("2. 최고 평균 점수 학생 검색");
        System.out.println("3. 최저 평균 점수 학생 검색");
        System.out.println("4. 전체 학생 평균 점수");
        System.out.println("5. 평균 점수 내림차순 학생 정렬");
        System.out.println("6. 학생 검색");
        System.out.println("7. 학생 추가");
        System.out.println("8. 종료");
        System.out.println("\n================================");
        System.out.print("원하는 메뉴를 선택하세요: ");

        int menu = scanner.nextInt();
        scanner.nextLine();

        switch (menu){
            case 1: printAllStudents();             break;
            case 2: searchHighestAverageStudent();  break;
            case 3: searchLowestAverageStudent();   break;
            case 4: getTotalStudentAverage();       break;
            case 5: sortStudentAverage();           break;
            case 6: searchStudent();                break;
            case 7: addStudent();                   break;
            case 8:
                System.out.println("프로그램을 종료합니다.");
                scanner.close();
                System.exit( status: 0);
            default:
                System.out.println("잘못된 메뉴를 선택하셨습니다. 다시 선택해주세요.");
        }
    }
}
```

✓ 최고/최저 평균점수 학생 검색 코드 수정

```java
//2. 최고 평균점수 학생 검색
private static void searchHighestAverageStudent(){
    if(studentMap.isEmpty()){ System.out.println("등록된 학생이 없습니다."); }

    double maxAvg=Double.MIN_VALUE;
    String maxKey = "";

    for (String id : studentMap.keySet()) {
        Student student = studentMap.get(id);
        double avg = student.getAverage();

        if (avg > maxAvg) {
            maxAvg = avg;
            maxKey = id;
        }
    }
    System.out.println("최고 평균 점수 학생: "
            +(studentMap.get(maxKey)).getName());
}
```

```java
//2. 최고 평균점수 학생 검색
private static void searchHighestAverageStudent(){
    if(studentMap.isEmpty()){ System.out.println("등록된 학생이 없습니다."); }

    System.out.println("최고 평균 점수 학생: "
      +studentMap.values().stream().max(Comparator.comparingDouble(Student::getAverage)).get());
}
```

✓ 전체 학생 평균 점수 출력

```
//4. 전체 학생 평균 점수
private static void getTotalStudentAverage(){
    if(studentMap.isEmpty()){ System.out.println("등록된 학생이 없습니다."); }

    System.out.println("전체 학생 평균 점수: "
      +studentMap.values().stream().mapToDouble(Student::getAverage).average().getAsDouble());
}
```

✓ 평균 점수 기준 내림차순으로 정렬

✓ 1) Student 클래스 수정 - Comparable 구현, compareTo 메서드 오버라이딩

```
public class Student implements Comparable<Student> {

    @Override
    public int compareTo(Student s){
        return (int)(this.getAverage() - s.getAverage());
    }
```

✓ 2) StudentManagement4 클래스에 해당 메서드 추가

```
//5. 평균 점수 내림차순 정렬
private static void sortStudentAverage(){
    if(studentMap.isEmpty()){ System.out.println("등록된 학생이 없습니다."); }

    System.out.println("평균 점수로 내림차순 정렬");
    studentMap.values().stream().sorted((a,b) -> b.compareTo(a) - a.compareTo(b)).forEach(System.out::println);
}
```

오름차순 : sorted()

```
studentMap.values().stream().sorted(Comparator.reverseOrder()).forEach(System.out::println);
```

추천