8253jang@daum.net
장희정

Stream API 10문제

☞ 알아야 할 메소드

- filter
- mapToDouble
- average
- max
- Comparator.comparingInt
- collect
- groupingBy
- joining
- allMatch
- anyMatch

메소드	설명	변환타입
filter(Predicate<T>)	특정 조건을 만족하는 요소만 필터링	Stream<T>
collect(Collector<T, A, R>)	결과를 리스트, 맵 등으로 변환	Collection or Map
mapToDouble(ToDoubleFunction<T>)	값을 double로 변환	DoubleStream
average()	평균값 계산	OptionalDouble
max(Comparator<T>)	최댓값 찾기	Optional<T>
Comparator.comparingInt(Function<T, Integer>)	정렬 기준 설정	Comparator<T>
groupingBy(Function<T, K>)	특정 기준으로 그룹화	Map<K, List<T>>
joining(CharSequence)	문자열을 연결하여 하나로 합침	String
allMatch(Predicate<T>)	모든 요소가 조건을 만족하는지 확인	boolean
anyMatch(Predicate<T>)	하나라도 조건을 만족하는지 확인	boolean

1. filter(Predicate<T>) - 특정 조건에 맞는 요소만 필터링

스트림에서 특정 조건을 만족하는 요소만 선택할 때 사용

Predicate<T> 조건식을 받아서 true 인 요소만 남김

```
List<String> names = List.of("Alice", "Bob", "Charlie", "David");
List<String> filteredNames = names.stream()
    .filter(name -> name.startsWith("A"))
    .collect(Collectors.toList());
```

```
System.out.println(filteredNames); // [Alice]
```

2. collect(Collector<T, A, R>) - 결과를 컬렉션 등으로 변환

스트림의 요소들을 모아서 리스트, 맵, 문자열 등으로 변환

```
List<String> nameList = names.stream()
    .collect(Collectors.toList());
```

3. mapToDouble(ToDoubleFunction<T>) - 값을 double 로 변환

map()과 비슷하지만 결과가 DoubleStream 이 됨

주로 평균이나 합계를 구할 때 사용

```
double avg = students.stream()
    .mapToDouble(Student::getScore)
    .average()
    .orElse(0.0);
```

4. average() - 평균값 계산

DoubleStream 에서 평균을 계산할 때 사용

결과는 OptionalDouble 이므로 .orElse(기본값)을 사용 가능

```
OptionalDouble avgScore = students.stream()
    .mapToDouble(Student::getScore)
    .average();

System.out.println(avgScore.orElse(0.0));
```

5. max(Comparator<T>) - 최댓값 찾기

Comparator<T>를 사용하여 최대값을 가진 요소를 찾음

결과는 Optional<T>로 반환됨

```
Optional<Student> topStudent = students.stream()
    .max(Comparator.comparingDouble(Student::getScore));

topStudent.ifPresent(System.out::println);
```

6. Comparator.comparingInt(Function<T, Integer>) - 정렬 기준 설정

정렬 시 정수 값 기준으로 비교할 때 사용

```
students.stream()
```

```
    .sorted(Comparator.comparingInt(Student::getAge))
    .forEach(System.out::println);
```

7. groupingBy(Function<T, K>) - 그룹화

특정 기준으로 그룹을 나눌 때 사용

결과는 Map<K, List<T>> 형태

```
Map<String, List<Student>> studentsByMajor = students.stream()
    .collect(Collectors.groupingBy(Student::getMajor));

System.out.println(studentsByMajor);
```

8. joining(CharSequence) - 문자열 연결

스트림의 요소들을 하나의 문자열로 합칠 때 사용

```
String studentNames = students.stream()
    .map(Student::getName)
    .collect(Collectors.joining(", "));

System.out.println(studentNames); // "Alice, Bob, Charlie"
```

9. allMatch(Predicate<T>) - 모든 요소가 조건을 만족하는지 확인

모든 요소가 주어진 조건을 만족하면 true 반환

하나라도 false 면 false 반환

```
boolean hasHighScorer = students.stream()
    .anyMatch(s -> s.getScore() >= 90);

System.out.println(hasHighScorer); // true or false
```

10. anyMatch(Predicate<T>) - 하나라도 조건을 만족하는지 확인

하나라도 조건을 만족하면 true 반환

모든 요소가 false 면 false 반환

```
boolean hasHighScorer = students.stream()
    .anyMatch(s -> s.getScore() >= 90);

System.out.println(hasHighScorer); // true or false
```

8253jang@daum.net
장희정

실습하기

☞ Student 정보

```
package stream.exam;
public class Student {
    private String name;
    private int age;
    private double score;
    private String major;

    public Student(String name, int age, double score, String major) {
        this.name = name;
        this.age = age;
        this.score = score;
        this.major = major;
    }

    public String getName() { return name; }
    public int getAge() { return age; }
    public double getScore() { return score; }
    public String getMajor() { return major; }

    @Override
    public String toString() {
        return "Student{name='" + name + "', age=" + age + ", score=" + score + ", major='" +
major + "'}";
    }
}
```

☞ 샘플 데이터

```
List<Student> students = Arrays.asList(
                new Student("희정", 22, 88.5, "Computer Science"),
                 new Student("가현", 24, 76.2, "Mathematics"),
                 new Student("찬범", 23, 92.3, "Computer Science"),
                 new Student("현솔", 25, 81.7, "Physics")
        );
```

8253jang@daum.net
장희정

☞ **문제**

```
/**
  1)
 * 평균 점수 구하기 (filter + mapToDouble + average)
 *  List<Student>에서 전공이 "Computer Science"인 학생들의 평균 점수를 구하시오.
 * */

 /**
 2)
 * 나이가 가장 많은 학생 찾기 (max + Comparator.comparingInt)
 *   List<Student>에서 나이가 가장 많은 학생의 정보를 출력하시오.
 * */

 /**
 3)
 * 점수가 80 점 이상인 학생 목록 추출 (filter + collect)
 *   List<Student>에서 점수가 80 점 이상인 학생들의 목록을 출력하시오.
 * */

 /**
 4)
 * 전공별 학생 수 구하기 (collect + groupingBy)
 *   List<Student>에서 전공별 학생 수를 구하시오.
 * */

 /**
 5)
 * 모든 학생 이름을 쉼표(,)로 연결 (map + collect + joining)
 *   List<Student>에서 모든 학생의 이름을 쉼표(,)로 연결하여 출력하시오.
 * */

 /**
 6)
 * 나이가 23 세 이상인 학생들의 이름 리스트 출력 (filter + map + collect)
 *  List<Student>에서 나이가 23 세 이상인 학생들의 이름을 리스트로 추출하시오.
 * */
```

```
 /**
 7)
 * 학생의 전공과 점수 목록을 출력 (map + collect)
 *  List<Student>에서 학생의 전공과 점수를 "전공: 점수" 형태의 리스트로 출력하시오.
 * */

 /**
 8)
 * 90 점 이상 학생의 전공(Major)을 대문자로 변환하여 출력 (filter + map + collect)
 *  List<Student>에서 90 점 이상인 학생들의 전공(Major)을 대문자로 변환하여 리스트로
출력하시오.
 * */

 /**
 9)
 * 모든 학생이 70 점 이상인지 확인 (allMatch)
 *  List<Student>의 모든 학생이 70 점 이상인지 확인하고, 그 결과를 출력하시오.
 * */

 /**
 10)
 * 전공이 'Computer Science'인 학생이 한 명이라도 있는지 확인(anyMatch)
 *  List<Student>에서 전공이 'Computer Science'인 학생이 한 명이라도 있는지 확인하고, 그
결과를 출력하시오.
 * */
```