8253jang@daum.net
장희정

Stream 에서 groupingBy 사용법

Collectors.groupingBy()는 Java Stream API 에서 데이터를 특정 기준으로 **그룹화**할 때 사용하는 메서드로 이터를 **Map<K, List<T>> 형태**로 변환해주며, **통계 계산**을 함께 수행할 수도 있다.

☞ 기본문법

```
Map<K, List<T>> result = list.stream()
    .collect(Collectors.groupingBy(기준함수));
```

기준함수 : 데이터를 어떤 기준으로 그룹화할지 결정하는 **람다식**
K : 그룹의 **키(key)**
T : 원본 리스트의 요소 타입

예) 학생을 전공별로 그룹화

```
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 name + " (" + score + ")";
        }
}
```

```
/**
 * 학생을 ~ 그룹화
 * */
public class GroupingByExample {
    public static void main(String[] args) {
        List<Student> students = Arrays.asList(
            new Student("희정", 22, 88.5, "Computer Science"),
            new Student("찬범", 24, 76.2, "Mathematics"),
            new Student("가현", 23, 92.3, "Computer Science"),
            new Student("현솔", 25, 81.7, "Physics"),
            new Student("현준", 21, 85.4, "Mathematics")
        );
        ////////////////////////////////////////////////////////////////////////////////////////
        /*
         * groupingBy(Student::getMajor) → 전공(major) 기준으로 그룹화
           같은 전공을 가진 학생들이 같은 그룹에 저장됨.
         * */
        System.out.println("-----1) 전공별로 학생을 그룹화----------------------");
        // 전공별로 학생을 그룹화
        Map<String, List<Student>> studentsByMajor = students.stream()
            .collect(Collectors.groupingBy(Student::getMajor));

        // 결과 출력
        /*studentsByMajor.forEach((new BiConsumer<String, List<Student>>() {
                        @Override
```

```
                        public void accept(String major, List<Student> studentList) {
                                System.out.println(major+" : " + studentList);
                        }}

                ));*/

    // 결과 출력
        studentsByMajor.forEach((major, studentList) ->
            System.out.println(major + ": " + studentList));

    ////////////////////////////////////////////////////////////////////////////////////////////
        /*
         * Collectors.counting() → 그룹 내 요소 수를 세는 연산
           Map<String, Long> 형태로 결과가 반환됨.
         * */

        System.out.println("₩n-----2)전공별 학생 수 계산 (counting)----------");
        Map<String, Long> majorCount = students.stream()
                .collect(Collectors.groupingBy(Student::getMajor, Collectors.counting()));

         System.out.println(majorCount);

////////////////////////////////////////////////////////////////////////////////////////////
        /**
         * Collectors.averagingDouble(Student::getScore) → 점수의 평균값을 계산
         * */
        System.out.println("₩n-----3) 전공별 평균 점수 계산 (mapping + averagingDouble))----------");
        Map<String, Double> averageScoresByMajor = students.stream()
                        .collect(Collectors.groupingBy(
                                Student::getMajor,
                                Collectors.averagingDouble(Student::getScore)
                        ));

          System.out.println(averageScoresByMajor);

    ////////////////////////////////////////////////////////////////////////////////////////////
    /*
    * Collectors.maxBy(Comparator.comparingDouble(Student::getScore)) → 최대값을 찾는 연산
        결과가 Optional<Student> 타입이라 .get()으로 값 추출
   * */
    System.out.println("₩n-----4) 전공별 최고 점수 학생 찾기 (maxBy)----------");
```

```
    Map<String, Optional<Student>> topStudentByMajor = students.stream()
            .collect(Collectors.groupingBy(
            Student::getMajor,
            Collectors.maxBy(Comparator.comparingDouble(Student::getScore))
           ));

        topStudentByMajor.forEach((major, student) -> System.out.println(major + ": " + student.get()));


  ////////////////////////////////////////////////////////////////////////////////////////////////
/*
* 첫 번째 groupingBy(Student::getMajor) → 전공별 그룹화
   두 번째 groupingBy(점수 기준) → 점수 등급별(A, B, C) 그룹화
  최종 결과는 Map<String, Map<String, List<Student>>> 형태
*/
  System.out.println("₩n-----5)다중 그룹화 (전공별, 성적 등급별 그룹)---------------");

  Map<String, Map<String, List<Student>>> grouped = students.stream()
            .collect(Collectors.groupingBy(
            Student::getMajor,
            Collectors.groupingBy(s -> {
              if (s.getScore() >= 90) return "A";
              else if (s.getScore() >= 80) return "B";
              else return "C";
            })
    ));

  grouped.forEach((a,b)->System.out.println(a+" => " + b));
   //System.out.println(grouped);
 }
}
```

정리

기능	코드
기본 그룹화	groupingBy(Student::getMajor)
그룹별 요소 개수	groupingBy(Student::getMajor, counting())
그룹별 평균값	groupingBy(Student::getMajor, averagingDouble(Student::getScore))
그룹별 최대값	groupingBy(Student::getMajor, maxBy(Comparator.comparingDouble(Student::getScore)))
다중 그룹화	groupingBy(Student::getMajor, groupingBy(기준))

8253jang@daum.net
장희정