CS109 2025Spring Assignment4 Group

> Designer: ZHU Yueming
>
> Junit: ZHANG Wencheng. Tester: YIN Chaoyue

What to Submit

Student.java

Group.java

GroupSystem.java

Introduction

This assignment is a basic exercise on classes and objects. The evaluation mechanism is different from previous assignments and it is based on junit5.8. We will provide a local junit test, which you can download through this link.

https://github.com/JAVA-Course-For-Sustech/materials_in_25Spring/blob/main/LocalTest.java

The topic is mainly used to simulate the function of finding ungrouped students and automatically grouping them when forming a project team.

Other Important parameters and Test cases

1. We ensure that in the test case, each student's number is unique.
2. We ensure that in the test case of **GroupSystem.group()** method, the grade of each student is unique.

Classes

1. class Student

- Do not modify or remove any methods or fields that have been already defined.
- You can add other methods or attributes that you think are necessary.

Fields:

`private String studentNumber;` The number of student.

`private char lab;` The lab of student. Throughout the assignment, we ensured that the lab characters in the test cases were only **26 uppercase English letters**.

`private int grade;` The current grade of student.

Methods:

- **constructor():**

  ```
  public Student(String studentNumber, char lab, int grade)
  ```

  In the construction method, each private fields needs to be initialized
- **Getter and Setter methods:**

  Add get and set methods to each private fields.
- **toString():**

  ```
  public String toString()
  ```

  Return the object of a string with a format below:

  ```
  [studentNumber] [lab] [grade]
  ```

  For example:

  ```
  124001 A 90
  124002 A 95
  ```

2. class Group

- Do not modify or remove any methods or fields that have been already defined.
- You can add other methods or attributes that you think are necessary.、

The team members can be two or one person.

Fields:

`private Student student1;` The first student in the group.

`private Student student2;` The second student in the group.

`private char lab;` The presentation time for the group.

Methods:

- **two constructors:**

  ```
  public Group(Student student1, Student student2, char lab)
  ```

  In the first construction method, each private fields needs to be initialized.

  ```
  public Group(Student student1, char lab)
  ```

In the second construction method, only initialize the private fields student1 and lab, and keep the private field student2 null.

- **Getter and Setter methods:**

  Add get and set methods to each private fields.

- **checkSameLab()**:

```
public boolean checkSameLab()
```

  Only when student1 and student2 are not null, and the labs of the two students are the same as the lab field in group, the return value will be true, otherwise the return value is false.

- **toString():**

```
public String toString()
```

  Return a string with a format below:

  If the second student is not null:

```
[lab]_[studentNumber]_[studentNumber]
```

  In the return value, of the two students, the studentNumber with the earlier alphabetical order is placed in front of the return value, and the studentNumber with the later alphabetical order is placed at the back.

  If the second student is null:

```
[lab]_[studentNumber1]_null
```

  For example:

```
public static void main(String[] args) {
        Student s1 = new Student("124001", 'A', 90);
        Student s2 = new Student("124002", 'A', 95);
        Student s3 = new Student("124003", 'B', 91);
        Group g1 = new Group(s2, s1, 'A');
        Group g2 = new Group(s3, 'B');

        System.out.println(g1);
        System.out.println(g2);
    }
```

  The result is:

```
A_124001_124002
B_124003_null
```

3. Class GroupSystem

Static Methods:

- **check()**

```java
public static Student[] check(Student[] students, Group[] groups)
```

This method is based on the two parameters: students and groups, excluding students with invalid groups and students who have not formed group, and counting the students who have not yet formed groups.

Parameters:

The **students** parameter represents the students who have taken the course. We ensure that there are no duplicates in the parameters passed in.
The **groups** parameter represents the students forming their own groups. We ensure that the students in the groups will appear in the students parameter.

What to return?

1. **Invalid groups** students:
   - A group with only one student.
   - If a student appears in different groups multiple times, then all students in these groups will be invalid.
   - And finally, the lab of the two students are not the same as the lab field in group.
2. **Students** haven't grouped.

   The student appearred in the paramter students, but not appearred in any parameter groups.
3. If there is **no students** will be returned, return **null**;

For example:

```java
public static void main(String[] args) {
        Student[] students = new Student[10];
        students[0] = new Student("12400", 'A', 95);
        students[1] = new Student("12401", 'A', 96);
        students[2] = new Student("12402", 'B', 92);
        students[3] = new Student("12403", 'B', 91);
        students[4] = new Student("12404", 'B', 90);
        students[5] = new Student("12405", 'B', 55);
        students[6] = new Student("12406", 'C', 45);
        students[7] = new Student("12407", 'C', 100);
        students[8] = new Student("12408", 'A', 99);
        students[9] = new Student("12409", 'A', 78);

        Group[] groups = new Group[6];
        groups[0] = new Group(students[0], students[9], 'A');
        groups[1] = new Group(students[1], students[8], 'B');
        groups[2] = new Group(students[2], students[4], 'B');
        groups[3] = new Group(students[2], students[5], 'B');
        groups[4] = new Group(students[3], 'B');
        groups[5] = new Group(students[6], students[7], 'C');

        Student[] nullGroups = GroupSystem.check(students,groups);
        for(Student s: nullGroups){
            System.out.println(s);
        }
    }
```

Result:

```
12401 A 96
12402 B 92
12403 B 91
12404 B 90
12405 B 55
12408 A 99
```

- **group()**

```
public static Group[] group(Student[] students)
```

This method groups the students in the parameters into groups using a certain way and returns the results of the grouping.

Parameter:

students: Students who need to form a team. We ensure that the student in the parameter students is unique.

How to group?

1. Cross-lab group is not allowed.
2. If multiple students are from the same lab, they will form groups in desending order of their grades. If the total number of students from the same lab is an odd number, the student with the lowest grade will form a single group.

Returns an array of groups formed according to the above requirements, and the length of the array must be consistent with the count of groups.

For example:

```
public static void main(String[] args) {
        Student[]  students= new Student[10];
        students[0] = new Student("12400",'A',96);
        students[1] = new Student("12401",'B',92);
        students[2] = new Student("12402",'A',91);
        students[3] = new Student("12403",'A',93);
        students[4] = new Student("12404",'H',99);
        students[5] = new Student("12405",'E',86);
        students[6] = new Student("12406",'A',76);
        students[7] = new Student("12407",'E',56);
        students[8] = new Student("12408",'B',36);
        students[9] = new Student("12409",'A',100);
        Group[] groups = GroupSystem.group(students);
        for(Group g: groups){
            System.out.println(g);
        }
    }
```

Result:

```
A_12400_12409
A_12402_12403
E_12405_12407
B_12401_12408
A_12406_null
H_12404_null
```