Assignment4

> Designer:
>
> Question1 and Question2: Wei Tian Yun Xi
>
> Question 3: ZHU Yueming
>
> Tester: WANG Zhuo, YIN Chaoyue

Question1: IPV4

Description

A valid IPv4 address is an IP address of the form $x_1.x_2.x_3.x_4$ Where $0 <= x_i <= 255$ and x_i is integer and it cannot start with zero. For example, $192.168.1.1$ and $192.168.1.0$ are valid IPv4 addresses, whereas $192.168.01.1$ and $192.168@1.1$, $192.00.2.1$, $192..2.255$ are invalid IPv4 addresses.

Input

The first line contains a string indicating the IPv4 address to be verified. In this section, any String may appear.

Output

Print "Yes" if the input string is a valid IPv4 address, otherwise print "No".

Input

```
172.16.254.1
```

Output

```
Yes
```

Input

```
256.256.256.256
```

Output

```
No
```

Question2: Palindrome

Description

A palindrome string is a special kind of string that is the same as the original after inversion. For example, $ABCBA$ is a palindrome. Given a string a, remove the other string b in a, you will get a string c. Ignore the case and remove all non-alphanumeric characters in c. Please determine whether the final string is a palindrome.

We guarantee that string a contains only one string b.

Input

The first line contains a string a. The second line contains a string b.

Output

Print "Yes" if the final string is a palindrome string, otherwise print "No".

Input

```
Ab!c1cbBba2A.
Bba
```

Output

```
Yes
```

Input

```
Ab!c1cbBba2A.
Bb
```

Output

```
No
```

Question3: Contribute Rate

We will use junit to test this question, and we have provided a local judge for you. You can download it here:

https://github.com/JAVA-Course-For-Sustech/materials_in_24fall/blob/main/A4Q3LocalTest.java

Submit:

Student.java

GradeSystem.java

Class 1: Student

This class mainly describes the data of students in group projects, such as the student's group score, group size, personal score, contribution rate, etc.

> For this class, do not modify or remove any fields and methods that have been already defined, however you can add other fields or methods if you need.

static field:

- **count**: It starts from 0, then it will be increased by 1 when a new student has been created. It is to record how many student objects have been created.

```
private static int count = 0;
```

static method:

- **calculatePersonalScore**: It is a static method to return personal score according to giving paramaters: group score, contribute rate and the group size.

```
public static double calculatePersonalScore(int groupScore, int rate, int groupSize)
```

  In this method, we can make sure all testcases are follow the rules below:

  - groupScore : is between 0 to 100.
  - rate: is between 0 to 100.
  - groupSize: can only be 2 or 3.

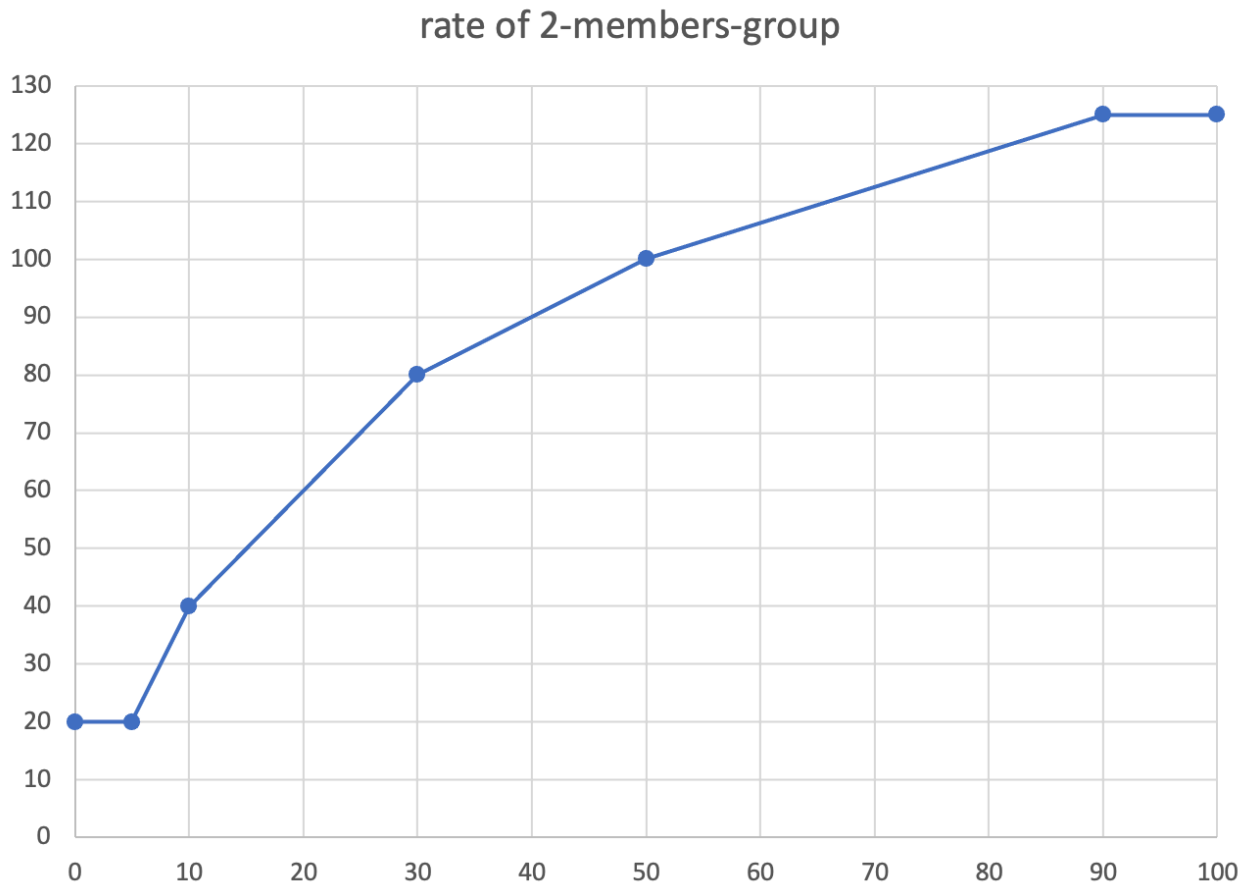
  The way to calculate personal score can be refered to the table below:

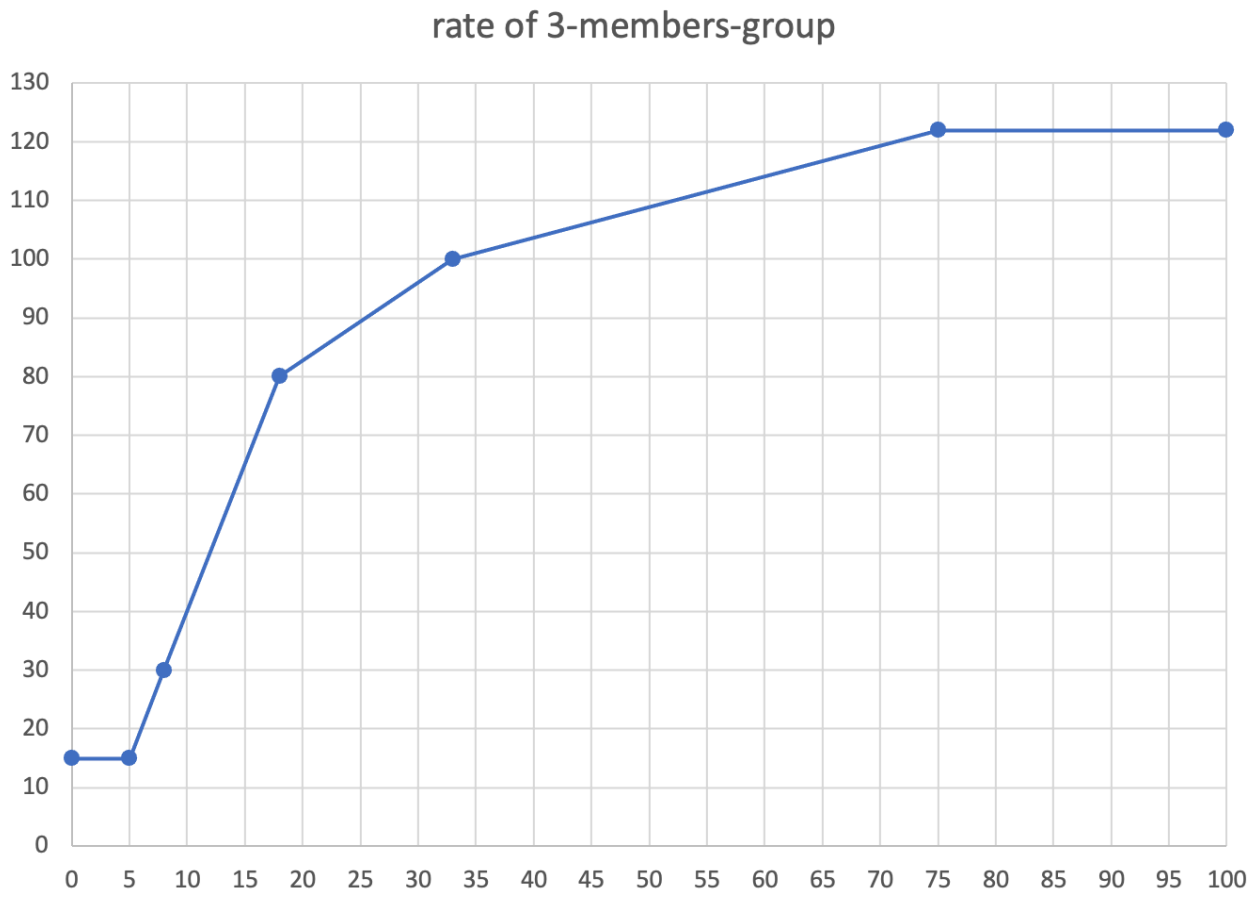
  > For example, if groupSize = 2, rate = 10, then the personal score would be 40% of the group score.

  > Another example, if groupSize = 2, rate = 8, we can define a value that is scope, and the scope can be (40-20)/(10-5) = 4 which means the percentage of score would be added by 4, for every 1 increase in rate. Then 8 is increased 3 from 5, so that the percentage of score woule be 20 + 3 * scope = 20 + 12 = 32. And finally, the personal score would be 32% of the group score.

Group Size	Rate of lower limit% (RL)	Rate of higher limit% (RH)	Lower percentage Limit % (L)	Higher percentage Limit% (H)
2	0	5	20	20
2	5	10	20	40
2	10	30	40	80
2	30	50	80	100
2	50	90	100	125
2	90	100	125	125
3	0	5	15	15
3	5	18	15	80
3	18	33	80	100
3	33	75	100	122
3	75	100	122	122

More specifically, the following two figures can better express the content described in the above table. In the figure, the horizontal axis represents the rate, and the vertical axis represents the percentage of the personal score corresponding to each rate in the group score.


rate of 2-members-group
130
120
110
100
90
80
70
60
50
40
30
20
10
0
0
10
20
30
40
50
60
70
80
90
100


rate of 3-members-group
130
120
110
100
90
80
70
60
50
40
30
20
10
0
0
5
10
15
20
25
30
35
40
45
50
55
60
65
70
75
80
85
90
95
100

Sample Test Data:

```
public static void main(String[] args) {
        System.out.println(Student.calculatePersonalScore(75,40,2));
        System.out.println(Student.calculatePersonalScore(75,50,2));
        System.out.println(Student.calculatePersonalScore(75,95,2));
        System.out.println(Student.calculatePersonalScore(75,100,2));
    }
```

Output:

```
67.5
75.0
93.75
93.75
```

Fields:

- **studentId**: The id of student. It starts from 1, and then it will be increased by 1 when a new student has been created. For example, when create a first student, the studentId is 1, when create a second student, the studentId is 2, etc.

  ```
  private int studentId;
  ```

- **groupSize**: The size of group. The default value can be zero, but we can make sure that among all testcases of passing parameters, the value of group size can only be 2 or 3.

  ```
  private int groupSize;
  ```

- **groupNumber**: The number of group. The default value can be (char)0, but we can make sure that among all testcases of passing parameters, the value of groupNumber can only be Capital letter.

  ```
  private char groupNumber;
  ```

- **groupScore**: The score of group. We can make sure that among all testcases of passing parameters, the value of groupScore can only in the range of 0 to 100.

  ```
  private int groupScore;
  ```

- **personalScore**: The score of personal. It can be larger than 100.

  ```
  private double personalScore;
  ```

- **rate**:The contribute rate of current student. We can make sure that among all testcases of passing parameters, the value of rate can only in the range of 0 to 100.

```
private int rate;//the contribute rate of current student
```

Member Methods:

- **Constructors**: Design two constructors like below, for all constructors:
  - You need given the original value of `studentId`.
  - Copy parameter values to corresponding fields.

```
public Student(int groupScore, int rate, char groupNumber)
```

and

```
public Student(int groupScore, int rate, int groupSize)
```

- **Getters and Setters**: Generate all getter and setter methods for all private fields.
- **updatePersonalScore()**: Update student's personal score according to his/her groupScore, rate and groupSize.

```
public void updatePersonalScore()
```

The calculation method is the same as that of static method **calculatePersonalScore**

- **toString()**: Design a toString() method to return the format below:

```
[studentId] group:[groupScore] personal:[personalScore] rate:[rate]
```

personalScore need to be rounded to integers.

For example:

```
 public static void main(String[] args) {
        Student s1 = new Student(80,25,3);
        Student s2 = new Student(80,35,3);
        Student s3 = new Student(80,45,3);
        s1.updatePersonalScore();
        s2.updatePersonalScore();
        s3.updatePersonalScore();
        System.out.println(s1);
        System.out.println(s2);
        System.out.println(s3);
    }
```

Output:

```
1 group:80 personal:71 rate:25
2 group:80 personal:81 rate:35
3 group:80 personal:85 rate:45
```

Class 2: GradeSystem

This class mainly contains some static methods, each of which will pass an array representing the collection of students. We will perform some operations to generate group size, personal scores, and standard group scores based on the parameters in the array.

static methods:

- **generateStudentGroupSize()**: This method is to generate the group size according to the parameter `students`.

  ```
  public static void generateStudentGroupSize(Student[] students)
  ```

  The parameter of `students` array we provide can satisfy the following conditions:

  1. All student objects are created through the construction method below, that is, the value of groupSize attribute is 0.

     ```
     public Student(int groupScore, int rate, char groupNumber)
     ```

  2. The groupNumber attribute of all student objects is only uppercase English letters.
  3. Each uppercase English letter will only appear 2 or 3 times in the students array.

     For example:

     ```
     public static void main(String[] args) {
             Student[] students = new Student[5];
             students[0] = new Student(80,35,'A');
             students[1] = new Student(60,45,'B');
             students[2] = new Student(80,55,'B');
             students[3] = new Student(70,40,'A');
             students[4] = new Student(70,25,'A');
             GradeSystem.generateStudentGroupSize(students);
             for (Student s: students) {
                 System.out.println(s.getGroupSize());
             }
         }
     ```

     Output:

```
3
2
2
3
3
```

- **standardizedScores()** :According to the parameter `students`, if students have same groupNumber but have different group scores, update the group score of those students to 60.

  ```
  public static void standardizedScores(Student[] students)
  ```

  The parameter of `students` array we provide can satisfy the following conditions:

  1. All student objects are created through the construction method below, that is, the value of groupSize attribute is 0.

     ```
     public Student(int groupScore, int rate, char groupNumber)
     ```

  2. The groupNumber attribute of all student objects is only uppercase English letters.
  3. Each uppercase English letter will only appear 2 or 3 times in the students array.

  For example:

  ```
  public static void main(String[] args) {
          Student[] students = new Student[7];
          students[0] = new Student(80,35,'A');
          students[1] = new Student(70,45,'B');
          students[2] = new Student(80,55,'B');
          students[3] = new Student(95,40,'C');
          students[4] = new Student(95,40,'A');
          students[5] = new Student(70,25,'A');
          students[6] = new Student(95,60,'C');
          GradeSystem.standardizedScores(students);
          for (Student s: students) {
              System.out.println(s);
          }
      }
  ```

  Output:

```
1 group:60 personal:0 rate:35
2 group:60 personal:0 rate:45
3 group:60 personal:0 rate:55
4 group:95 personal:0 rate:40
5 group:60 personal:0 rate:40
6 group:60 personal:0 rate:25
7 group:95 personal:0 rate:60
```

- **generatePersonalScore()**: This method is to generate the personal score of all students in the parameter `students`.

  The parameter of `students` array we provide can satisfy the condition that the group size of all student objects in parameter `students` array is 2 or 3.

  However, we also have the testcase that all student objects are created through the construction method `public Student(int groupScore, int rate, char groupNumber)`, and then execute the static method `generateStudentGroupSize()`, and finally exercise this method.

```
public static void generatePersonalScore(Student[] students)
```

  For example:

```
public static void main(String[] args) {
        Student[] students = new Student[5];
        students[0] = new Student(50,35,3);
        students[1] = new Student(50,35,3);
        students[2] = new Student(50,30,3);
        students[3] = new Student(50,40,2);
        students[4] = new Student(50,60,2);
        GradeSystem.generatePersonalScore(students);
        for (Student s: students) {
            System.out.println(s);
        }
    }
```

  Output:

```
1 group:50 personal:51 rate:35
2 group:50 personal:51 rate:35
3 group:50 personal:48 rate:30
4 group:50 personal:45 rate:40
5 group:50 personal:53 rate:60
```