Rajalakshmi Engineering College

Name: Haresh R
Email: 241901031@rajalakshmi.edu.in
Roll no:
Phone: null
Branch: REC
Department: CSE (CS) - Section 1
Batch: 2028
Degree: B.E - CSE (CS)

Scan to verify results

2024_28_III_OOPS Using Java Lab

REC_2028_OOPS using Java_Week 10_CY

Attempt : 1
Total Mark : 40
Marks Obtained : 40

Section 1 : COD

1. Problem Statement

Tony is an e-learning platform administrator, he oversees the user ratings for various online courses offered in the platform.

To enhance user experience, you should assist him in utilizing a HashMap to store course ratings given by learners. Regularly, he analyzes this data to identify the highest and lowest-rated courses, enabling targeted improvements and ensuring the quality of the educational content. This process assists in maintaining a competitive and engaging online learning environment for the users.

Input Format

The input consists of a string representing the course name followed by a double value representing the course's rating, in separate lines.

The input is terminated by entering "done".

Output Format

The first line of output prints the string "Highest Rated Course: " followed by the highest-rated course.

The second line prints the string "Lowest Rated Course: " followed by the lowest-rated courses.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: DSA
4.0
OOPS
4.2
C
3.2
done

Output: Highest Rated Course: OOPS
Lowest Rated Course: C

Answer

```
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;

import java.util.*;

class CourseAnalyzer {
   public Map<String, String>
identifyHighestAndLowestRatedCourses(Map<String, Double> courseRatings) {
      String highestCourse = "";
      String lowestCourse = "";
      double highestRating = Double.NEGATIVE_INFINITY;
      double lowestRating = Double.POSITIVE_INFINITY;

      for (Map.Entry<String, Double> entry : courseRatings.entrySet()) {
```

```
            String course = entry.getKey();
            double rating = entry.getValue();

            if (rating > highestRating) {
                highestRating = rating;
                highestCourse = course;
            }
            if (rating < lowestRating) {
                lowestRating = rating;
                lowestCourse = course;
            }
        }

        Map<String, String> result = new HashMap<>();
        result.put("highest", highestCourse);
        result.put("lowest", lowestCourse);
        return result;
    }
}

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Map<String, Double> courseRatings = new HashMap<>();

        while (true) {
            String courseName = scanner.nextLine();
            if (courseName.equalsIgnoreCase("done")) {
                break;
            }
            double rating = Double.parseDouble(scanner.nextLine().trim());
            courseRatings.put(courseName, rating);
        }

        CourseAnalyzer analyzer = new CourseAnalyzer();
        Map<String, String> result =
analyzer.identifyHighestAndLowestRatedCourses(courseRatings);

        System.out.printf("Highest Rated Course: %s\n", result.get("highest"));
        System.out.printf("Lowest Rated Course: %s", result.get("lowest"));

        scanner.close();
    }
```

}

Status : Correct ***Marks : 10/10***

2. Problem Statement

Aryan is developing a voting system for a college election. Each vote is recorded as an entry in an array, where every student's vote is represented by a candidate's ID. Since it's a majority-rule election, the winner is the candidate who receives more than n/2 votes, where n is the total number of votes cast.

To quickly determine the winner, Aryan decides to use a HashMap to count the occurrences of each vote and identify the candidate who has received more than half of the total votes.

Example

Input

7

2 2 1 2 2 2 3

Output

2

Explanation

The votes are: 2, 2, 1, 2, 2, 3, 2

Count of each candidate:

2 appears 5 times1 appears once3 appears once

The majority element is the one that appears more than N/2 times. Since 7/2 = 3.5, a number must appear at least 4 times to be the majority.

The number 2 appears 5 times, which is greater than 3.5, so the output is 2.

Input Format

The first line contains an integer N representing the number of votes cast.

The second line contains N space-separated integers representing the votes, where each integer corresponds to a candidate.

Output Format

The output prints an integer representing the majority element (the candidate who received more than N/2 votes).

If no such candidate exists, print -1.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 7
2 2 1 2 2 2 3
Output: 2

Answer

```
import java.util.HashMap;
import java.util.Scanner;

import java.util.*;

class MajorityElementFinder {

    public static int findMajorityElement(int[] arr) {
        Map<Integer, Integer> freq = new HashMap<>();
        int n = arr.length;


        for (int vote : arr) {
            freq.put(vote, freq.getOrDefault(vote, 0) + 1);
        }


        for (Map.Entry<Integer, Integer> entry : freq.entrySet()) {
            if (entry.getValue() > n / 2) {
                return entry.getKey();
            }
```

```
        }

        return -1;
    }
}

class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int N = scanner.nextInt();
        int[] arr = new int[N];

        for (int i = 0; i < N; i++) {
            arr[i] = scanner.nextInt();
        }

        int result = MajorityElementFinder.findMajorityElement(arr);
        System.out.println(result);

        scanner.close();
    }
}
```

Status : Correct ***Marks : 10/10***

3. Problem Statement

Bob wants to develop a score-tracking application for a gaming tournament. Each player's score is stored in a HashMap with the player's name as the key and the score as the value.

Write a program to assist Bob that takes user input to enter player scores, calculates the maximum score from the HashMap, and prints the player with the highest score.

Input Format

The input consists of strings representing player details in the format "playerName:score".

The input is terminated by entering "done".

Output Format

The output displays a string, representing the player's name who scored the maximum.

If the value is not numeric, print "Invalid input".

If any special characters other than ':' are given, print "Invalid format".

Refer to the sample output for formatting specifications.

Sample Test Case

Input: Alice:15
Bob:56
done
Output: Bob

Answer

```
import java.util.*;

import java.util.*;

class ScoreTracker {

   Map<String, Integer> scoreMap = new HashMap<>();

   public boolean processInput(String input) {


      if (!input.matches("[A-Za-z]+:\\w+")) {


         if (!input.contains(":")) {
            System.out.println("Invalid format");
            return false;
         }
      }


      if (input.indexOf(':') != input.lastIndexOf(':')) {
```

```
            System.out.println("Invalid format");
            return false;
        }

        String[] parts = input.split(":");
        if (parts.length != 2) {
            System.out.println("Invalid format");
            return false;
        }

        String player = parts[0];
        String scoreStr = parts[1];


        if (!player.matches("[A-Za-z]+")) {
            System.out.println("Invalid format");
            return false;
        }


        if (!scoreStr.matches("\\d+")) {
            System.out.println("Invalid input");
            return false;
        }

        int score = Integer.parseInt(scoreStr);

        scoreMap.put(player, score);
        return true;
    }

    public String findTopPlayer() {
        String topPlayer = "";
        int maxScore = Integer.MIN_VALUE;

        for (Map.Entry<String, Integer> entry : scoreMap.entrySet()) {
            if (entry.getValue() > maxScore) {
                maxScore = entry.getValue();
                topPlayer = entry.getKey();
            }
        }
        return topPlayer;
```

```
    }
}
public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        ScoreTracker tracker = new ScoreTracker();
        boolean validInput = true;

        while (true) {
            String input = scanner.nextLine();

            if (input.toLowerCase().equals("done")) {
                break;
            }

            if (!tracker.processInput(input)) {
                validInput = false;
                break;
            }
        }

        if (validInput && !tracker.scoreMap.isEmpty()) {
            System.out.println(tracker.findTopPlayer());
        }

        scanner.close();
    }
}
```

Status : Correct ***Marks : 10/10***

4. Problem Statement

A college professor wants to keep track of students who attend classes. Each student has a unique roll number and their attendance count increases every time they attend a class. The system should allow adding a student, marking their attendance, and displaying all students with their total attendance.

Your task is to implement a Java program using TreeSet to maintain

students in sorted order of roll numbers and track their attendance count.

Operations:

A roll_no name Add a student with roll number and name (if not already added).M roll_no Mark attendance for the student with the given roll number (increase their count by 1).D Display all students in ascending order of roll number along with their attendance count.

Input Format

The first line contains an integer N - the number of students.

The next N lines contain one of the following commands:

A roll_no name

M roll_no

D

- A (Add) Adds a new student with a unique roll number and name.
- M (Mark) Increases attendance count for the given roll number.
- D (Display) Prints all students in ascending order of roll number.

Output Format

For D, output prints each student's roll number, name, and attendance count in ascending order of roll number.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 5
A 101 Alice
A 102 Bob
M 101
M 101
D

Output: 101 Alice 2
102 Bob 0

Answer

```
import java.util.*;

class Student implements Comparable<Student> {
   int rollNo;
   String name;
   int attendance;

   public Student(int rollNo, String name) {
      this.rollNo = rollNo;
      this.name = name;
      this.attendance = 0;
   }

   @Override
   public int compareTo(Student other) {
      return this.rollNo - other.rollNo;
   }
}

public class Main {
   public static void main(String[] args) {

      Scanner sc = new Scanner(System.in);
      int N = Integer.parseInt(sc.nextLine());

      TreeSet<Student> students = new TreeSet<>();

      while (N-- > 0) {
         String line = sc.nextLine().trim();
         String[] parts = line.split(" ");

         String cmd = parts[0];

         if (cmd.equals("A")) {
            int roll = Integer.parseInt(parts[1]);
            String name = parts[2];


           boolean exists = false;
            for (Student s : students) {
               if (s.rollNo == roll) {
```

```
                exists = true;
                break;
            }
        }


        if (!exists) {
            students.add(new Student(roll, name));
        }
    }
    else if (cmd.equals("M")) {
        int roll = Integer.parseInt(parts[1]);

        for (Student s : students) {
            if (s.rollNo == roll) {
                s.attendance++;
                break;
            }
        }
    }
    else if (cmd.equals("D")) {
        for (Student s : students) {
            System.out.println(s.rollNo + " " + s.name + " " + s.attendance);
        }
    }
  }

  sc.close();
 }
}
```

Status : Correct **Marks : 10/10**