Rajalakshmi Engineering College

Name: Lakshitha K
Email: 241801132@rajalakshmi.edu.in
Roll no: 241801132
Phone: 6381920328
Branch: REC
Department: AI & DS - Section 3
Batch: 2028
Degree: B.E - AI & DS

Scan to verify results

2024_28_III_OOPS Using Java Lab

REC_2028_OOPS using Java_Week 10_PAH

Attempt : 1
Total Mark : 30
Marks Obtained : 25

Section 1 : Coding

1. Problem Statement

Riya is building a calendar event scheduler where each event is stored in chronological order using a TreeMap. The key represents the event time in 24-hour format (HH:MM), and the value is the event description.

She wants the system to:

Automatically sort events by time.Avoid duplicate time entries — if a duplicate time is entered, ignore the new entry.Print all scheduled events in order.

Implement this logic using a class named EventManager.

Input Format

The first line of the input contains an integer n, representing the number of events.

The next n lines each contain a string in the format: "HH:MM Description"

(Example: 09:00 TeamMeeting).

Output Format

The first line of the output prints "Scheduled Events:"

The next k lines print each event in the format: "HH:MM - Description"

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 5
09:00 TeamMeeting
13:30 LunchBreak
11:00 ProjectUpdate
09:00 Standup
15:00 ClientCall

Output: Scheduled Events:
09:00 - TeamMeeting
11:00 - ProjectUpdate
13:30 - LunchBreak
15:00 - ClientCall

Answer

```
// You are using Java
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        // Read number of events
        int n = Integer.parseInt(sc.nextLine());

        // Whitelist constraint: 1 ≤ n ≤ 50
        if (n < 1 || n > 50) {
            System.out.println("Invalid number of events.");
            return;
        }

        TreeMap<String, String> eventMap = new TreeMap<>();

        for (int i = 0; i < n; i++) {
            String line = sc.nextLine();

            // Validate format: must contain exactly one space separating time and
description
            if (!line.contains(" ") || line.indexOf(" ") != line.lastIndexOf(" ")) {
                System.out.println("Invalid input format.");
                return;
            }

            String[] parts = line.split(" ");
            String time = parts[0];
            String description = parts[1];

            // Whitelist validation: HH:MM format and description without spaces
            if (!time.matches("\\d{2}:\\d{2}") || description.contains(" ")) {
                System.out.println("Invalid input format.");
                return;
            }

            // Avoid duplicate time entries
            eventMap.putIfAbsent(time, description);
        }

        // Output scheduled events
        System.out.println("Scheduled Events:");
        for (Map.Entry<String, String> entry : eventMap.entrySet()) {
            System.out.println(entry.getKey() + " - " + entry.getValue());
        }

        sc.close();
    }
}
```

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

2. Problem Statement

Sarah is working on a spam detection system that analyzes incoming messages for unique patterns. Spammers often use repetitive character sequences, making it important to identify the first non-repeating character in a message.

Given a string, Sarah needs to determine the first character that appears only once. If all characters repeat, the system should return -1.

She decides to use a HashMap to efficiently track character frequencies and find the solution.

Input Format

The first line contains an integer N representing , the length of the string.

The second line contains a string of N lowercase English letters (a-z).

Output Format

The output prints a character representing the first non-repeating character. If none exist, print -1.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 10
abacabadac

Output: d

Answer

```
// You are using Java
import java.util.*;

public class Main {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
```

```
        // Read length of the string
        int N = Integer.parseInt(sc.nextLine());

        // Whitelist constraint: 1 ≤ N ≤ 100
        if (N < 1 || N > 100) {
            System.out.println("Invalid string length.");
            return;
        }

        String input = sc.nextLine();

        // Validate string length and character set
        if (input.length() != N || !input.matches("[a-z]+")) {
            System.out.println("Invalid input format.");
            return;
        }

        // Track character frequencies
        HashMap<Character, Integer> freqMap = new HashMap<>();

        for (char ch : input.toCharArray()) {
            freqMap.put(ch, freqMap.getOrDefault(ch, 0) + 1);
        }

        // Find first non-repeating character
        char result = '-';
        for (char ch : input.toCharArray()) {
            if (freqMap.get(ch) == 1) {
                result = ch;
                break;
            }
        }

        // Output result
        if (result == '-') {
            System.out.println("-1");
        } else {
            System.out.println(result);
        }

        sc.close();
    }
```

```
}
```

Status : Partially correct ***Marks : 5/10***

3. Problem Statement

A university maintains a list of student records and wants to store them in a sorted manner based on their GPA. If two students have the same GPA, they should be further sorted by their name in lexicographical order. Implement a program that uses a TreeSet to store student records and ensures unique student IDs.

Input Format

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

The next N lines contain details of each student in the format: "StudentID Name GPA"

- StudentID (Integer) - A unique identifier.
- Name (String) - The student's name (can contain spaces).
- GPA (Double) - The Grade Point Average.

Output Format

The output prints the list of students in ascending order of GPA.

If two students have the same GPA, sort them by name.

Print details in the format: "StudentID Name GPA" in the output, GPA is rounded to two decimal places.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 5
101 John 8.5
102 Alice 9.1
103 Bob 8.5

```
104 Zoe 7.3
105 Charlie 9.1
Output: 104 Zoe 7.30
103 Bob 8.50
101 John 8.50
102 Alice 9.10
105 Charlie 9.10
```

Answer

```
// You are using Java
import java.util.*;
import java.text.DecimalFormat;

class Student implements Comparable<Student> {
   int id;
   String name;
   double gpa;

   public Student(int id, String name, double gpa) {
      this.id = id;
      this.name = name;
      this.gpa = gpa;
   }

   // Sort by GPA, then by name lexicographically
   @Override
   public int compareTo(Student other) {
      if (Double.compare(this.gpa, other.gpa) != 0) {
         return Double.compare(this.gpa, other.gpa);
      }
      return this.name.compareTo(other.name);
   }

   @Override
   public String toString() {
      DecimalFormat df = new DecimalFormat("0.00");
      return id + " " + name + " " + df.format(gpa);
   }

   // Ensure uniqueness by StudentID
   @Override
   public boolean equals(Object obj) {
        if (this == obj) return true;
        if (!(obj instanceof Student)) return false;
        Student other = (Student) obj;
        return this.id == other.id;
    }

    @Override
    public int hashCode() {
        return Integer.hashCode(id);
    }
}

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        int N = Integer.parseInt(sc.nextLine());

        // Whitelist constraint: 1 ≤ N ≤ 1000
        if (N < 1 || N > 1000) {
            System.out.println("Invalid number of students.");
            return;
        }

        TreeSet<Student> studentSet = new TreeSet<>();
        Set<Integer> idSet = new HashSet<>();

        for (int i = 0; i < N; i++) {
            String line = sc.nextLine();
            String[] parts = line.trim().split(" ", 3);

            if (parts.length != 3) {
                System.out.println("Invalid input format.");
                return;
            }

            try {
                int id = Integer.parseInt(parts[0]);
                String name = parts[1];
                double gpa = Double.parseDouble(parts[2]);

                // Whitelist validations
                if (id < 1 || id > 1_000_000 || name.length() < 1 || name.length() > 100 ||
gpa < 0.0 || gpa > 10.0 || !name.matches("[a-zA-Z ]+")) {
                    System.out.println("Invalid input values.");
                    return;
                }

                if (idSet.contains(id)) {
                    continue; // Ignore duplicate IDs
                }

                studentSet.add(new Student(id, name, gpa));
                idSet.add(id);

            } catch (NumberFormatException e) {
                System.out.println("Invalid number format.");
                return;
            }
        }

        // Output
        for (Student s : studentSet) {
            System.out.println(s);
        }

        sc.close();
    }
}
```

Status : Correct *Marks : 10/10*