Rajalakshmi Engineering College

Name: Prince Rohith
Email: 240701399@rajalakshmi.edu.in
Roll no: 240701399
Phone: 6369941431
Branch: REC
Department: CSE - Section 5
Batch: 2028
Degree: B.E - CSE

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

David is managing an employee database where each employee has a unique ID, name, and department. He wants to ensure that duplicate employee IDs are not added to the system. Implement a Java program that allows adding employees to the system, displaying all employees, and checking if an employee exists based on the given ID.

Implement a class EmployeeDatabase that contains a HashSet to store employee records. The Employee class should be a user-defined object containing employee details. The main class should handle user operations and interact with the EmployeeDatabase class.

Input Format

The first line contains an integer n representing the number of employees to be added.

The next n lines follow, each containing:

1. An integer employee_id
2. A string name
3. A string department

The next line contains an integer m representing the number of queries.

The next m lines follow, each containing an employee ID to check for existence.

Output Format

The output prints a list of all employees added in the format:

"ID: <employee_id>, Name: <name>, Department: <department>"

For each query, output "Employee exists" if the ID is found, otherwise "Employee not found".

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 3
101 John IT
102 Alice HR
103 Bob Finance
2
101
104

Output: ID: 101, Name: John, Department: IT
ID: 102, Name: Alice, Department: HR
ID: 103, Name: Bob, Department: Finance
Employee exists
Employee not found

Answer

```
import java.util.*;

class Employee {
```

```
    private final int id;
    private final String name;
    private final String department;

    Employee(int id, String name, String department) {
        this.id = id;
        this.name = name;
        this.department = department;
    }

    int getId() { return id; }

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

    @Override
    public boolean equals(Object obj) {
        if (this == obj) return true;
        if (!(obj instanceof Employee)) return false;
        return this.id == ((Employee) obj).id;
    }

    @Override
    public String toString() {
        return "ID: " + id + ", Name: " + name + ", Department: " + department;
    }
}

class EmployeeDatabase {
    private final HashSet<Employee> set = new HashSet<>();

    boolean addEmployee(int id, String name, String department) {
        return set.add(new Employee(id, name, department));
    }

    void displayEmployees() {
        for (Employee e : set) {
            System.out.println(e);
        }
    }
```

```
    boolean checkEmployee(int id) {
        return set.contains(new Employee(id, "", ""));
    }
}

class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        EmployeeDatabase db = new EmployeeDatabase();
        int n = sc.nextInt();
        for (int i = 0; i < n; i++) {
            int id = sc.nextInt();
            String name = sc.next();
            String department = sc.next();
            db.addEmployee(id, name, department);
        }
        db.displayEmployees();
        int m = sc.nextInt();
        for (int i = 0; i < m; i++) {
            int id = sc.nextInt();
            if (db.checkEmployee(id))
                System.out.println("Employee exists");
            else
                System.out.println("Employee not found");
        }
        sc.close();
    }
}
```

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

2. Problem Statement

A linguist named Meera is classifying a list of words based on their first character. She wants to store words grouped by their starting letter using a TreeMap so that the groups appear in sorted order of characters (i.e., 'a' to 'z'). For each letter, all words starting with that letter should be stored in the order they appear.

Implement the logic inside a class named WordClassifier using the

TreeMap<Character, List<String>> collection.

Input Format

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

The next n lines each contain a word.

Output Format

The first line of the output prints: "Grouped Words by Starting Letter:"

The next lines print each character key and its list of words in the format:

"letter: word1 word2 word3...

..."

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 5
dog
deer
cat
cow
camel
Output: Grouped Words by Starting Letter:
c: cat cow camel
d: dog deer

Answer

```
import java.util.*;

class WordClassifier {
    public void classifyWords(List<String> words) {
        TreeMap<Character, List<String>> map = new TreeMap<>();

        for (String w : words) {
```

```
            if (w == null || w.isEmpty()) continue;
            char key = Character.toLowerCase(w.charAt(0));
            map.computeIfAbsent(key, k -> new ArrayList<>()).add(w);
        }

        System.out.println("Grouped Words by Starting Letter:");
        for (Map.Entry<Character, List<String>> e : map.entrySet()) {
            StringBuilder line = new StringBuilder();
            line.append(e.getKey()).append(": ");
            List<String> list = e.getValue();
            for (int i = 0; i < list.size(); i++) {
                line.append(list.get(i));
                if (i < list.size() - 1) line.append(' ');
            }
            line.append(' ');
            System.out.println(line.toString().trim());
        }
    }
}

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

        List<String> words = new ArrayList<>();
        for (int i = 0; i < n; i++) {
            words.add(sc.nextLine());
        }

        WordClassifier classifier = new WordClassifier();
        classifier.classifyWords(words);
    }
}
```

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

3. 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.Scanner;
import java.util.TreeSet;

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 Integer.compare(this.rollNo, other.rollNo);
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj) return true;
        if (!(obj instanceof Student)) return false;
        return this.rollNo == ((Student) obj).rollNo;
    }

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

    @Override
    public String toString() {
        return rollNo + " " + name + " " + attendance;
    }
```

```
}

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

        int N = Integer.parseInt(scanner.nextLine().trim());

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

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

            String command = parts[0];

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

                Student newStudent = new Student(rollNo, name);

                Student existing = findByRollNo(students, rollNo);
                if (existing != null) {
                    students.remove(existing);
                }
                students.add(newStudent);

            } else if (command.equals("M")) {
                int rollNo = Integer.parseInt(parts[1]);

                Student student = findByRollNo(students, rollNo);
                if (student != null) {
                    student.attendance++;
                }


            } else if (command.equals("D")) {
                for (Student s : students) {
                    System.out.println(s);
                }
            }
```

```
        }

        scanner.close();
    }

    private static Student findByRollNo(TreeSet<Student> set, int rollNo) {
        for (Student s : set) {
            if (s.rollNo == rollNo) {
                return s;
            }
        }
        return null;
    }
}
```

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

4. 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.*;

class ScoreTracker {
    HashMap<String, Integer> scoreMap = new HashMap<>();
    public boolean processInput(String input) {
        if (input.equalsIgnoreCase("done")) {
            return false;
        }

        if (!input.contains(":") || input.chars().filter(ch -> ch == ':').count() != 1) {
            System.out.println("Invalid format");
            System.exit(0);
        }

        String[] parts = input.split(":");
        if (parts.length != 2) {
            System.out.println("Invalid format");
            System.exit(0);
        }

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

        if (!name.matches("[A-Za-z]+")) {
            System.out.println("Invalid format");
            System.exit(0);
        }

        if (!scoreStr.matches("\\d+")) {
```

```
            System.out.println("Invalid input");
            System.exit(0);
        }

        int score = Integer.parseInt(scoreStr);

        if (name.length() > 20 || score < 1 || score > 100) {
            System.out.println("Invalid input");
            System.exit(0);
        }

        scoreMap.put(name, 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***