

Rajalakshmi Engineering College

Name: KRITHESHWARAN R
Email: 241901049@rajalakshmi.edu.in
Roll no: 241901049
Phone: 9843565002
Branch: REC
Department: CSE (CS) - Section 2
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

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 {
```

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

```
    boolean processInput(String input) {
```

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

```
        String[] parts = input.split(":");
```

```
        String playerName = parts[0].trim();
```

```
        String scoreStr = parts[1].trim();
```

```
        try {
```

```
            int score = Integer.parseInt(scoreStr);
```

```
            if (score < 1 || score > 100) {
```

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

```
        scoreMap.put(playerName, score);
        return true;
    } catch (NumberFormatException e) {
        System.out.println("Invalid input");
        return false;
    }
}
```

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

```
    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

2. Problem Statement

Arjun is working on a program that checks if one set of numbers is a subset of another. If Set B is a subset of Set A, the program should print "YES" followed by the sorted elements of Set B. If Set B is not a subset of Set A, the program should print "NO" followed by the average of all elements from both sets combined, rounded to two decimal places.

Implement a class Solution with the required method to perform the subset check using TreeSet in Java.

Input Format

The first line contains an integer n - the number of elements in Set A.

The second line contains n space-separated integers - the elements of Set A.

The third line contains an integer m - the number of elements in Set B.

The fourth line contains m space-separated integers - the elements of Set B.

Output Format

If Set B is a subset of Set A, print "YES" followed by the sorted values of Set B.

Otherwise, print "NO" followed by the average of all numbers in both sets (rounded to two decimal places).

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 5

1 2 3 4 5

3

2 3 5

Output: YES 2 3 5

Answer

```
import java.util.*;
```

```
class Solution {  
    public static void checkSubset(TreeSet<Integer> setA, TreeSet<Integer> setB,  
int totalElements, double sum) {  
        if (setA.containsAll(setB)) {  
            System.out.print("YES ");  
            for (int num : setB) {  
                System.out.print(num + " ");  
            }  
            System.out.println();  
        } else {  
            double average = sum / totalElements;  
            System.out.printf("NO %.2f%n", average);  
        }  
    }  
}
```

```
class Main {  
    public static void main(String[] args) {  
        Scanner sc = new Scanner(System.in);  
        int n = sc.nextInt();  
        TreeSet<Integer> setA = new TreeSet<>();  
        long sum = 0;  
        for (int i = 0; i < n; i++) {  
            int num = sc.nextInt();  
            setA.add(num);  
            sum += num;  
        }  
        int m = sc.nextInt();  
        TreeSet<Integer> setB = new TreeSet<>();  
        for (int i = 0; i < m; i++) {  
            int num = sc.nextInt();  
            setB.add(num);  
        }  
    }  
}
```

```
        sum += num;
    }
    Solution.checkSubset(setA, setB, n + m, sum);
    sc.close();
}
}
```

Status : Correct

Marks : 10/10

3. 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 {  
    int employeeId;  
    String name, department;
```

```
    public Employee(int employeeId, String name, String department) {  
        this.employeeId = employeeId;  
        this.name = name;  
        this.department = department;  
    }
```

```
    public int hashCode() {  
        return Objects.hash(employeeId);  
    }
```

```
    public boolean equals(Object obj) {
```

```

    if (this == obj) return true;
    if (obj == null || getClass() != obj.getClass()) return false;
    Employee e = (Employee) obj;
    return this.employeeId == e.employeeId;
}

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

class EmployeeDatabase {
    HashSet<Employee> employees = new HashSet<>();

    public void addEmployee(int id, String name, String department) {
        employees.add(new Employee(id, name, department));
    }

    public void displayEmployees() {
        for (Employee e : employees) {
            System.out.println(e);
        }
    }

    public boolean checkEmployee(int id) {
        return employees.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

4. 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 word : words) {
```

```
            char initial = word.charAt(0);
```

```
            if (!map.containsKey(initial)) {
```

```
                map.put(initial, new ArrayList<>());
```

```
            }
```

```
            map.get(initial).add(word);
```

```
        }
```

```
        System.out.println("Grouped Words by Starting Letter:");
```

```
        for (Map.Entry<Character, List<String>> entry : map.entrySet()) {
```

```
            System.out.print(entry.getKey() + ": ");
```

```
            for (String word : entry.getValue()) {
```

```
                System.out.print(word + " ");
```

```
            }
```

```
            System.out.println();
```

```
        }
```

```
    }
```

```
}
```

```
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