

# Rajalakshmi Engineering College

Name: Mohamed Yahya A  
Email: 240701325@rajalakshmi.edu.in  
Roll no: 240701325  
Phone: 9600561844  
Branch: REC  
Department: CSE - Section 9  
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

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

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 times 1 appears once 3 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;

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

        for (int num : arr) {
            countMap.put(num, countMap.getOrDefault(num, 0) + 1);
        }
        for (int key : countMap.keySet()) {
            if (countMap.get(key) > n / 2) {
                return key;
            }
        }
    }
}
```

```

        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

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

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