

# Rajalakshmi Engineering College

Name: srivatsen s  
Email: 240701534@rajalakshmi.edu.in  
Roll no: 2116240701534  
Phone: 9042122714  
Branch: REC  
Department: CSE - Section 7  
Batch: 2028  
Degree: B.E - CSE

Scan to verify results



## 2024\_28\_III\_OOPS Using Java Lab

### **REC\_2028\_OOPS using Java\_Week 10\_MCQ**

Attempt : 1  
Total Mark : 15  
Marks Obtained : 15

#### **Section 1 : MCQ**

1. What will be the output of the following code?

```
import java.util.*;
class Main {
    public static void main(String[] args) {
        HashMap<String, Integer> map = new HashMap<>();
        map.put("X", 10);
        map.put("Y", 20);
        map.put("Z", 30);
        map.remove("Y");
        System.out.println(map);
    }
}
```

**Answer**

{X=10, Z=30}

**Status : Correct**

**Marks : 1/1**

2. What happens if two keys have the same hash code in a HashMap?

**Answer**

A linked list is used to store values with the same hash

**Status : Correct**

**Marks : 1/1**

3. Which method retrieves the lowest key in a TreeMap?

**Answer**

firstKey()

**Status : Correct**

**Marks : 1/1**

4. What will happen if you add elements in descending order in a TreeSet?

**Answer**

They are sorted in ascending order

**Status : Correct**

**Marks : 1/1**

5. Which statement is true about HashSet and TreeSet?

**Answer**

TreeSet provides sorted elements

**Status : Correct**

**Marks : 1/1**

6. Which of the following is true about TreeMap?

**Answer**

It maintains natural ordering

**Status : Correct**

**Marks : 1/1**

7. What will be the output of the following code?

```
import java.util.*;
class Main {
    public static void main(String[] args) {
        HashMap<String, Integer> map = new HashMap<>();
        map.put("A", 1);
        map.put("B", 2);
        map.put("C", 3);
        System.out.println(map.containsKey("B"));
    }
}
```

**Answer**

true

**Status : Correct**

**Marks : 1/1**

8. How does HashSet check for duplicate elements?

**Answer**

Using equals() and hashCode()

**Status : Correct**

**Marks : 1/1**

9. Which of the following allows null keys in Java?

**Answer**

HashMap

**Status : Correct**

**Marks : 1/1**

10. Which of the following is true about HashMap?

**Answer**

It is not synchronized

**Status : Correct**

**Marks : 1/1**

11. What will happen if you add a null element to a TreeSet?

**Answer**

An exception occurs

**Status : Correct**

**Marks : 1/1**

12. What happens when you add duplicate elements to a HashSet?

**Answer**

The duplicate is ignored

**Status : Correct**

**Marks : 1/1**

13. What is the time complexity of retrieving an element from a HashSet?

**Answer**

O(1)

**Status : Correct**

**Marks : 1/1**

14. Which method removes all elements from a Set?

**Answer**

clear()

**Status : Correct**

**Marks : 1/1**

15. What will be the output of the following code?

```
import java.util.*;
class Main {
    public static void main(String[] args) {
        HashMap<String, String> map = new HashMap<>();
        map.put("A", "Apple");
        map.put("B", "Banana");
        map.put("C", "Cherry");
```

```
        map.replace("B", "Blueberry");
        System.out.println(map);
    }
}
```

**Answer**

{A=Apple, B=Blueberry, C=Cherry}

**Status :** Correct

**Marks :** 1/1

# Rajalakshmi Engineering College

Name: srivatsen s  
Email: 240701534@rajalakshmi.edu.in  
Roll no: 2116240701534  
Phone: 9042122714  
Branch: REC  
Department: CSE - Section 7  
Batch: 2028  
Degree: B.E - CSE

Scan to verify results



## 2024\_28\_III\_OOPS Using Java Lab

### 2028\_REC\_OOPS using Java\_Week 10\_Q1

Attempt : 1  
Total Mark : 10  
Marks Obtained : 10

#### **Section 1 : COD**

##### **1. Problem Statement**

A city traffic management system needs to track vehicles entering a toll booth. Each vehicle is uniquely identified by its registration number. The system should allow adding vehicles to a record, ensuring that no duplicate registration numbers exist. The vehicles should be stored in a HashSet, which does not guarantee any specific order.

Your task is to implement a program using a HashSet that allows adding vehicle details and displaying the records.

##### ***Input Format***

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

The next N lines contain details of each vehicle in the format: "RegNumber

OwnerName VehicleType"

1. RegNumber (String) - A unique registration number (Alphanumeric).
2. OwnerName (String) - The name of the vehicle owner.
3. VehicleType (String, Car, Bike, or Truck) - The type of vehicle.

If a vehicle with the same registration number is already present, ignore the duplicate entry.

### ***Output Format***

The output prints the unique vehicle records in any order (since HashSet does not maintain order).

Output format: "RegNumber OwnerName VehicleType"

Refer to the sample output for formatting specifications.

### ***Sample Test Case***

Input: 5

KA01AB1234 John Car

MH02CD5678 Alice Bike

DL03EF9012 Bob Truck

TN04GH3456 Mike Car

KA01AB1234 John Car

Output: TN04GH3456 Mike Car

KA01AB1234 John Car

MH02CD5678 Alice Bike

DL03EF9012 Bob Truck

### ***Answer***

```
import java.util.*;  
class Vehicle {  
    String regNumber;  
    String ownerName;  
    String vehicleType;  
    private static HashSet<Vehicle> vehicleSet = new HashSet<>();  
    public Vehicle(String regNumber, String ownerName, String vehicleType) {  
        this.regNumber = regNumber;  
        this.ownerName = ownerName;
```

```

        this.vehicleType = vehicleType;
    }
    public static void addVehicle(String regNumber, String ownerName, String
vehicleType) {
        vehicleSet.add(new Vehicle(regNumber, ownerName, vehicleType));
    }
    public static void displayVehicles() {
        for (Vehicle v : vehicleSet) {
            System.out.println(v);
        }
    }
    public boolean equals(Object obj) {
        if (this == obj) return true;
        if (obj == null || getClass() != obj.getClass()) return false;
        Vehicle vehicle = (Vehicle) obj;
        return regNumber.equals(vehicle.regNumber);
    }
    public int hashCode() {
        return Objects.hash(regNumber);
    }
    public String toString() {
        return regNumber + " " + ownerName + " " + vehicleType;
    }
}
class TollBoothSystem {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        sc.nextLine();
        for (int i = 0; i < n; i++) {
            String regNumber = sc.next();
            String ownerName = sc.next();
            String vehicleType = sc.next();
            Vehicle.addVehicle(regNumber, ownerName, vehicleType);
        }
        Vehicle.displayVehicles();
        sc.close();
    }
}

```

**Status :** Correct

**Marks :** 10/10

# Rajalakshmi Engineering College

Name: srivatsen s  
Email: 240701534@rajalakshmi.edu.in  
Roll no: 2116240701534  
Phone: 9042122714  
Branch: REC  
Department: CSE - Section 7  
Batch: 2028  
Degree: B.E - CSE

Scan to verify results



## 2024\_28\_III\_OOPS Using Java Lab

### 2028\_REC\_OOPS using Java\_Week 10\_Q2

Attempt : 1  
Total Mark : 10  
Marks Obtained : 10

#### **Section 1 : COD**

##### **1. Problem Statement**

John is organizing a fruit festival, and the quantities of various fruits are stored in a HashMap where fruit names are keys and quantities are values.

Help him develop a program to find the total quantity of fruits for the festival by summing up the values in the HashMap.

##### ***Input Format***

The input consists of fruit quantities in the format 'fruitName:quantity', where fruitName is the name of the fruit(a string), and quantity is a double value representing the quantity.

The input is terminated by entering "done".

##### ***Output Format***

The output prints a double value, representing the sum of values in the HashMap, rounded off to two decimal places.

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

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

Refer to the sample output for formatting specifications.

### **Sample Test Case**

Input: Banana:15.2

Orange:56.3

Mango:47.3

done

Output: 118.80

### **Answer**

```
import java.util.Scanner;
import java.util.Map;
import java.util.HashMap;
class ValueProcessor {
    public static Map<String, Double> readValues(Scanner scanner) {
        Map<String, Double> valueMap = new HashMap<>();
        while (true) {
            String input = scanner.nextLine();
            if (input.toLowerCase().equals("done")) {
                break;
            }
            String[] pair = input.split(":");
            if (pair.length == 2) {
                String key = pair[0].trim();
                try {
                    double value = Double.parseDouble(pair[1].trim());
                    valueMap.put(key, value);
                } catch (NumberFormatException e) {
                    System.out.println("Invalid input");
                    return null;
                }
            } else {

```

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

public static double calculateSum(Map<String, Double> valueMap) {
    double sum = 0;
    for (double value : valueMap.values()) {
        sum += value;
    }
    return sum;
}
class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Map<String, Double> valueMap = ValueProcessor.readValues(scanner);
        if (valueMap != null) {
            double sum = ValueProcessor.calculateSum(valueMap);
            System.out.printf("%.2f\n", sum);
        }
        scanner.close();
    }
}
```

**Status :** Correct

**Marks :** 10/10

# Rajalakshmi Engineering College

Name: srivatsen s  
Email: 240701534@rajalakshmi.edu.in  
Roll no: 2116240701534  
Phone: 9042122714  
Branch: REC  
Department: CSE - Section 7  
Batch: 2028  
Degree: B.E - CSE

Scan to verify results



## 2024\_28\_III\_OOPS Using Java Lab

### 2028\_REC\_OOPS using Java\_Week 10\_Q3

Attempt : 1  
Total Mark : 10  
Marks Obtained : 10

#### **Section 1 : COD**

##### **1. Problem Statement**

Priya is analyzing encrypted messages in a research project. She wants to analyze the frequency of each character in a given paragraph. The characters should be stored in a TreeMap so that the output is sorted in ascending order of characters automatically.

You are required to build a Java program that:

Uses a TreeMap<Character, Integer> to count how many times each character appears in the message. Ignores spaces and considers only alphabets (case-sensitive). Outputs the frequencies of characters in sorted order.

You must use a TreeMap in the class named MessageAnalyzer.

#### ***Input Format***

The first line of input contains an integer n, the number of lines in the message.

The next n lines each contain a string (the encrypted message line).

### **Output Format**

The first line of output prints: "Character Frequency:"

Then print each character and its frequency in the format: "<character>: <count>"

Refer to the sample output for formatting specifications.

### **Sample Test Case**

Input: 2

Hello World

Java

Output: Character Frequency:

H: 1

J: 1

W: 1

a: 2

d: 1

e: 1

l: 3

o: 2

r: 1

v: 1

### **Answer**

```
import java.util.*;
class MessageAnalyzer {
    public void analyzeMessageFrequency(List<String> lines) {
        TreeMap<Character, Integer> frequencyMap = new TreeMap<>();

        for (String line : lines) {
            for (char ch : line.toCharArray()) {
                if (Character.isLetter(ch)) {
                    frequencyMap.put(ch, frequencyMap.getOrDefault(ch, 0) + 1);
                }
            }
        }
    }
}
```

```
        }
        System.out.println("Character Frequency:");
        for (Map.Entry<Character, Integer> entry : frequencyMap.entrySet()) {
            System.out.println(entry.getKey() + ": " + entry.getValue());
        }
    }
}

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = Integer.parseInt(sc.nextLine());
        List<String> lines = new ArrayList<>();
        for (int i = 0; i < n; i++) {
            lines.add(sc.nextLine());
        }

        MessageAnalyzer analyzer = new MessageAnalyzer();
        analyzer.analyzeMessageFrequency(lines);
    }
}
```

**Status :** Correct

**Marks :** 10/10

# Rajalakshmi Engineering College

Name: srivatsen s  
Email: 240701534@rajalakshmi.edu.in  
Roll no: 2116240701534  
Phone: 9042122714  
Branch: REC  
Department: CSE - Section 7  
Batch: 2028  
Degree: B.E - CSE

Scan to verify results



## 2024\_28\_III\_OOPS Using Java Lab

### 2028\_REC\_OOPS using Java\_Week 10\_Q4

Attempt : 1  
Total Mark : 10  
Marks Obtained : 10

#### **Section 1 : COD**

##### **1. Problem Statement**

In a ticket reservation system, you store the available seat numbers in a TreeSet. Users input their desired seat number, and the program checks whether the chosen seat is available.

Using a TreeSet ensures quick and efficient verification of seat availability, ensuring a smooth and organized ticket booking process.

##### ***Input Format***

The first line of input contains a single integer n, representing the number of available seats.

The second line contains n space-separated integers, representing the available seat numbers.

The third line contains an integer m, representing the seat number that needs to be searched.

#### ***Output Format***

The output displays "[m] is present!" if the given seat is available. Otherwise, it displays "[m] is not present!"

Refer to the sample output for the formatting specifications.

#### ***Sample Test Case***

Input: 4

2 4 5 6

5

Output: 5 is present!

#### ***Answer***

```
import java.util.Set;
import java.util.TreeSet;
import java.util.Scanner;
class NumberChecker {
    private Set<Integer> numberSet;

    public NumberChecker(Set<Integer> numberSet) {
        this.numberSet = numberSet;
    }

    public void addNumbers(int[] numbers) {
        for (int number : numbers) {
            numberSet.add(number);
        }
    }

    public String checkNumber(int number) {
        return numberSet.contains(number) ? number + " is present!" : number + " is
not present!";
    }
}
class Main {
    public static void main(String[] args) {
```

```
Scanner scanner = new Scanner(System.in);
int numberOfElements = scanner.nextInt();
int[] numbers = new int[numberOfElements];

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

int elementToCheck = scanner.nextInt();
scanner.close();

Set<Integer> numberSet = new TreeSet<>();
NumberChecker numberChecker = new NumberChecker(numberSet);
numberChecker.addNumbers(numbers);

System.out.println(numberChecker.checkNumber(elementToCheck));
}
```

**Status :** Correct

**Marks :** 10/10

# Rajalakshmi Engineering College

Name: srivatsen s  
Email: 240701534@rajalakshmi.edu.in  
Roll no: 2116240701534  
Phone: 9042122714  
Branch: REC  
Department: CSE - Section 7  
Batch: 2028  
Degree: B.E - CSE

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 : 30

#### **Section 1 : Coding**

##### **1. 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**

```
import java.util.*;
class Student implements Comparable<Student> {
    int studentID;
    String name;
    double gpa;

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

    public int compareTo(Student other) {
```

```

        if (this.gpa != other.gpa) {
            return Double.compare(this.gpa, other.gpa);
        }
        return this.name.compareTo(other.name);
    }

    public String toString() {
        return studentID + " " + name + " " + String.format("%.2f", gpa);
    }
}

class UniversityRecords {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        sc.nextLine();
        TreeSet<Student> studentSet = new TreeSet<>();
        for (int i = 0; i < n; i++) {
            int id = sc.nextInt();
            String name = sc.next();
            double gpa = sc.nextDouble();
            studentSet.add(new Student(id, name, gpa));
        }
        for (Student s : studentSet) {
            System.out.println(s);
        }
        sc.close();
    }
}

```

**Status :** Correct

**Marks :** 10/10

## 2. 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***

```
import java.util.*;
class EventManager {
    TreeMap<String, String> schedule;
    public EventManager() {
```

```

        schedule = new TreeMap<>();
    }

    public void addEvent(String time, String description) {
        if (!schedule.containsKey(time)) {
            schedule.put(time, description);
        }
    }

    public void printSchedule() {
        System.out.println("Scheduled Events:");
        for (Map.Entry<String, String> entry : schedule.entrySet()) {
            System.out.println(entry.getKey() + " - " + entry.getValue());
        }
    }
}

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

        EventManager manager = new EventManager();

        for (int i = 0; i < n; i++) {
            String line = sc.nextLine();
            int spaceIndex = line.indexOf(' ');
            String time = line.substring(0, spaceIndex);
            String desc = line.substring(spaceIndex + 1);
            manager.addEvent(time, desc);
        }

        manager.printSchedule();
    }
}

```

**Status :** Correct

**Marks :** 10/10

### 3. 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***

```
import java.util.*;  
class NonRepeatingCharacterFinder {  
  
    public char findFirstNonRepeatingCharacter(String str) {  
        HashMap<Character, Integer> charCount = new HashMap<>();  
  
        for (char ch : str.toCharArray()) {  
            charCount.put(ch, charCount.getOrDefault(ch, 0) + 1);  
        }  
  
        for (char ch : str.toCharArray()) {
```

```
        if (charCount.get(ch) == 1) {
            return ch;
        }
    }
    return '\0';
}
}
class FirstNonRepeatingCharacter {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        int N = sc.nextInt();
        String str = sc.next();

        NonRepeatingCharacterFinder finder = new NonRepeatingCharacterFinder();
        char result = finder.findFirstNonRepeatingCharacter(str);

        if (result == '\0') {
            System.out.println(-1);
        } else {
            System.out.println(result);
        }

        sc.close();
    }
}
```

**Status : Correct**

**Marks : 10/10**

# Rajalakshmi Engineering College

Name: srivatsen s  
Email: 240701534@rajalakshmi.edu.in  
Roll no: 2116240701534  
Phone: 9042122714  
Branch: REC  
Department: CSE - Section 7  
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**

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;

class CourseAnalyzer {
    public Map<String, String>
identifyHighestAndLowestRatedCourses(Map<String, Double> courseRatings) {
    double highestRating = Double.MIN_VALUE;
    double lowestRating = Double.MAX_VALUE;
    String highestRatedCourse = "";
    String lowestRatedCourse = "";

    for (Map.Entry<String, Double> entry : courseRatings.entrySet()) {
        String course = entry.getKey();
        double rating = entry.getValue();
```

```
if (rating > highestRating) {  
    highestRating = rating;  
    highestRatedCourse = course;  
}  
if (rating < lowestRating) {  
    lowestRating = rating;  
    lowestRatedCourse = course;  
}  
}  
  
Map<String, String> result = new HashMap<>();  
result.put("highest", highestRatedCourse);  
result.put("lowest", lowestRatedCourse);  
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();  
    }  
}
```

## 2. Problem Statement

The city library maintains a record of books available for lending. Each book is uniquely identified by its ISBN number, along with its title and author. The librarian wants to efficiently store and manage these records, ensuring books can be listed in the order they were added.

Your task is to implement a Library Management System using HashSet where:

The librarian adds books with ISBN, title, and author. The librarian can remove books by providing an ISBN. Finally, the librarian displays the available books in the order they were added.

Implement a class Library that will handle these operations. The main function should manage user input and interact with the Library class accordingly.

### ***Input Format***

The first line contains an integer n – the number of books to be added.

The next n lines contain three values: ISBN (integer), Title (string without spaces), and Author (string without spaces).

1. An integer employee\_id
2. A string title
3. A string author name

The next line contains an integer m – the number of books to be removed.

The next m lines follow, each contains an ISBN number to remove.

### ***Output Format***

The output prints a list of books available in the library after performing all operations in the format:

"ISBN: <isbn>, Title: <title>, Author: <author>"

If no books remain, print: "No books available"

Refer to the sample output for formatting specifications.

### **Sample Test Case**

Input: 3

1234 JavaCompleteGuide JohnDoe

5678 PythonBasics JaneDoe

9012 DataStructures AliceSmith

1

5679

Output: ISBN: 1234, Title: JavaCompleteGuide, Author: JohnDoe

ISBN: 9012, Title: DataStructures, Author: AliceSmith

ISBN: 5678, Title: PythonBasics, Author: JaneDoe

### **Answer**

```
import java.util.*;  
  
class Book {  
    int isbn;  
    String title, author;  
  
    public Book(int isbn, String title, String author) {  
        this.isbn = isbn;  
        this.title = title;  
        this.author = author;  
    }  
  
    public boolean equals(Object obj) {  
        if (this == obj) return true;  
        if (obj == null || getClass() != obj.getClass()) return false;  
        Book book = (Book) obj;  
        return isbn == book.isbn;  
    }  
  
    public int hashCode() {  
        return Objects.hash(isbn);  
    }  
}
```

```
}

class Library {
    HashSet<Book> books = new HashSet<>();

    void addBook(int isbn, String title, String author) {
        books.add(new Book(isbn, title, author));
    }

    void removeBook(int isbn) {
        books.removeIf(book -> book.isbn == isbn);
    }

    void displayBooks() {
        if (books.isEmpty()) {
            System.out.println("No books available");
        } else {
            for (Book book : books) {
                System.out.println("ISBN: " + book.isbn + ", Title: " + book.title + ",
Author: " + book.author);
            }
        }
    }
}

class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Library library = new Library();
        int n = sc.nextInt();
        for (int i = 0; i < n; i++) {
            int isbn = sc.nextInt();
            String title = sc.next();
            String author = sc.next();
            library.addBook(isbn, title, author);
        }
        int m = sc.nextInt();
        for (int i = 0; i < m; i++) {
            int isbn = sc.nextInt();
            library.removeBook(isbn);
        }
        library.displayBooks();
        sc.close();
    }
}
```

```
}
```

Status : Correct

Marks : 10/10

### 3. 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

#### 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;  
    }  
}
```

```
public void markAttendance() {
    this.attendance++;
}

public int compareTo(Student s) {
    return Integer.compare(this.rollNo, s.rollNo);
}

public boolean equals(Object obj) {
    if (this == obj) return true;
    if (obj == null || getClass() != obj.getClass()) return false;
    Student student = (Student) obj;
    return rollNo == student.rollNo;
}

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

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

```
class AttendanceTracker {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        sc.nextLine();
        TreeSet<Student> students = new TreeSet<>();
        for (int i = 0; i < n; i++) {
            String[] command = sc.nextLine().split(" ");
            String operation = command[0];

            if (operation.equals("A")) {
                int rollNo = Integer.parseInt(command[1]);
                String name = command[2];
                students.add(new Student(rollNo, name));
            }
            else if (operation.equals("M")) {
                int rollNo = Integer.parseInt(command[1]);
                for (Student s : students) {
```

```
        if (s.rollNo == rollNo) {
            s.markAttendance();
            break;
        }
    }
else if (operation.equals("D")) {
    for (Student s : students) {
        System.out.println(s);
    }
}
sc.close();
}
```

**Status :** Correct

**Marks :** 10/10

# Rajalakshmi Engineering College

Name: srivatsen s  
Email: 240701534@rajalakshmi.edu.in  
Roll no: 2116240701534  
Phone: 9042122714  
Branch: REC  
Department: CSE - Section 7  
Batch: 2028  
Degree: B.E - CSE

Scan to verify results



## 2024\_28\_III\_OOPS Using Java Lab

### REC\_2028\_OOPS using Java\_Week 11

Attempt : 1  
Total Mark : 20  
Marks Obtained : 20

#### **Section 1 : Project**

##### **1. Problem Statement**

In ABC Corporation, employee records are stored in a database.

To efficiently manage employee details using Java and JDBC, you are tasked with building an Employee Management System that supports the following functionalities:

Adding a new employee

Updating an employee's salary

Viewing an employee's details

Displaying all employees

You are given two files:

## File 1: Employee.java (POJO Class)

This class represents the Employee entity.

An Employee contains the following details:

### Field Description

employeeId Unique Employee ID (Integer)

name Employee Name (String)

department Employee Department (String)

salary Employee Salary (Double)

Students must write code in the marked area:

```
class Employee {  
    private int employeeId;  
    private String name;  
    private String department;  
    private double salary;  
  
    public Employee() {}  
  
    public Employee(int employeeId, String name, String department, double  
salary) {  
        // write your code here  
    }  
  
    // Include getters and setters  
}
```

Expected in this part:

Assign parameter values to instance variables inside the constructor.

Add getters and setters for all attributes.

File 2: EmployeeDAO.java (Data Access Layer)

This class handles all database operations using JDBC.

Students must complete the missing JDBC logic in the following methods:

```
class EmployeeDAO {  
  
    public void addEmployee(Connection conn, Employee employee) throws  
SQLException {  
        // write your code here  
    }  
  
    public void updateSalary(Connection conn, int employeeId, double  
newSalary) throws SQLException {  
        // write your code here  
    }  
  
    public void deleteEmployee(Connection conn, int employeeId) throws  
SQLException {  
        // write your code here  
    }  
  
    public Employee viewEmployeeRecord(Connection conn, int employeeId)  
throws SQLException {  
        // write your code here  
    }  
  
    public List<Employee> displayAllEmployees(Connection conn) throws  
SQLException {  
        // write your code here  
    }  
}
```

```
private Employee mapToEmployee(ResultSet rs) throws SQLException {  
    return new Employee(  
        // write your code here  
    );  
}  
}
```

Expected in this part:

Write SQL queries for INSERT, UPDATE, DELETE, SELECT.

Execute queries using PreparedStatement or Statement.

Map ResultSet rows to Employee objects using mapToEmployee().

Return a List<Employee> where required.

The system should connect to a MySQL database using the following default credentials:

DB URL: jdbc:mysql://localhost/ri\_db  
Username: test  
Password: test123

The employees table has already been created with the following structure:

#### ***Input Format***

The first line of input consists of an integer choice, representing the operation to be performed:

(1 for Add Employee, 2 for Update Salary, 3 for View Employee Record, 4 for Display All Employees, 5 for Exit)

For choice 1 (Add Employee):

1. The second line consists of an integer employee\_id.
2. The third line consists of a string name.
3. The fourth line consists of a string department.
4. The fifth line consists of a double salary (must be at least 30000).

For choice 2 (Update Salary):

1. The second line consists of an integer employee\_id.
2. The third line consists of a double new\_salary (must be at least 30000).

For choice 3 (View Employee Record):

1. The second line consists of an integer employee\_id.

For choice 4 (Display All Employees).

For choice 5 (Exit).

***Output Format***

For choice 1 (Add Employee),

1. Print "Employee added successfully" if the employee was added.

For choice 2 (Update Salary),

1. Print "Salary updated successfully" if the salary update was successful.
2. Print "Employee not found." if the specified employee ID does not exist.
3. Print "Salary must be at least 30000." if the provided salary is below the minimum.

For choice 3 (View Employee Record),

1. Display the employee details in the format:
2. ID: [employee\_id] | Name: [name] | Department: [department] | Salary: [salary]
3. Print "Employee not found." if the specified employee ID does not exist.

For choice 4 (Display All Employees),

1. Display each employee on a new line in the format:
2. ID | Name | Department | Salary

For choice 5 (Exit),

1. Print "Exiting Employee Management System."

For invalid input:

1. Print "Invalid choice. Please try again."

#### **Sample Test Case**

Input: 1

101

Alice Johnson

Engineering

31000.75

4

6

5

Output: Employee added successfully

ID | Name | Department | Salary

101 | Alice Johnson | Engineering | 31000.75

Invalid choice. Please try again.

Exiting Employee Management System.

#### **Answer**

```
import java.sql.*;  
import java.util.Scanner;  
  
class Employee {  
    private int employeeId;  
    private String name;  
    private String department;  
    private double salary;
```

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

// Getters and Setters
public int getEmployeeId() { return employeeId; }
public void setEmployeeId(int employeeId) { this.employeeId = employeeId; }

public String getName() { return name; }
public void setName(String name) { this.name = name; }

public String getDepartment() { return department; }
public void setDepartment(String department) { this.department = department; }

public double getSalary() { return salary; }
public void setSalary(double salary) { this.salary = salary; }
}
```

```
class EmployeeManagementSystem {

    // Add Employee
    public static void addEmployee(Connection conn, Scanner scanner) {
        int employeeId = scanner.nextInt();
        scanner.nextLine(); // Consume newline
        String name = scanner.nextLine();
        String department = scanner.nextLine();
        double salary = scanner.nextDouble();

        if (salary < 30000) {
            System.out.println("Salary must be at least 30000.");
            return;
        }

        // Create an Employee POJO object
        Employee employee = new Employee(employeeId, name, department,
                                         salary);
```

```
String insertQuery = "INSERT INTO employees (employee_id, name, department, salary) VALUES (?, ?, ?, ?);  
try (PreparedStatement stmt = conn.prepareStatement(insertQuery)) {  
    stmt.setInt(1, employee.getEmployeeId());  
    stmt.setString(2, employee.getName());  
    stmt.setString(3, employee.getDepartment());  
    stmt.setDouble(4, employee.getSalary());  
  
    int rowsInserted = stmt.executeUpdate();  
    System.out.println(rowsInserted > 0 ? "Employee added successfully" :  
        "Failed to add employee.");  
} catch (SQLException e) {  
    System.out.println("Error adding employee: " + e.getMessage());  
}  
  
// Update Salary  
public static void updateSalary(Connection conn, Scanner scanner) {  
    int employeeId = scanner.nextInt();  
    double newSalary = scanner.nextDouble();  
  
    if (newSalary < 30000) {  
        System.out.println("Salary must be at least 30000.");  
        return;  
    }  
  
    String updateQuery = "UPDATE employees SET salary = ? WHERE  
employee_id = ?";  
    try (PreparedStatement stmt = conn.prepareStatement(updateQuery)) {  
        stmt.setDouble(1, newSalary);  
        stmt.setInt(2, employeeId);  
  
        int rowsUpdated = stmt.executeUpdate();  
        System.out.println(rowsUpdated > 0 ? "Salary updated successfully" :  
            "Employee not found.");  
    } catch (SQLException e) {  
        System.out.println("Error updating salary: " + e.getMessage());  
    }  
  
    // View Employee Record
```

```
public static void viewEmployeeRecord(Connection conn, Scanner scanner) {
    int employeeId = scanner.nextInt();
    String selectQuery = "SELECT * FROM employees WHERE employee_id = ?";

    try (PreparedStatement stmt = conn.prepareStatement(selectQuery)) {
        stmt.setInt(1, employeeId);
        ResultSet rs = stmt.executeQuery();

        if (rs.next()) {
            Employee employee = new Employee(
                rs.getInt("employee_id"),
                rs.getString("name"),
                rs.getString("department"),
                rs.getDouble("salary")
            );
            System.out.printf("ID: %d | Name: %s | Department: %s | Salary: %.2f%n",
                employee.getEmployeeId(),
                employee.getName(),
                employee.getDepartment(),
                employee.getSalary());
        } else {
            System.out.println("Employee not found.");
        }
    } catch (SQLException e) {
        System.out.println("Error retrieving employee record: " + e.getMessage());
    }
}

// Display All Employees
public static void displayAllEmployees(Connection conn) {
    String displayQuery = "SELECT * FROM employees";

    try (Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery(displayQuery)) {

        System.out.println("ID | Name | Department | Salary");
        while (rs.next()) {
            Employee employee = new Employee(
                rs.getInt("employee_id"),
                rs.getString("name"),
                rs.getString("department"),
                rs.getDouble("salary"))
        }
    }
}
```

```
        );
        System.out.printf("%d | %s | %s | %.2f%n",
            employee.getEmployeeId(),
            employee.getName(),
            employee.getDepartment(),
            employee.getSalary());
    }
} catch (SQLException e) {
    System.out.println("Error displaying employees: " + e.getMessage());
}
}

public static void main(String[] args) {
    String url = "jdbc:mysql://localhost/ri_db";
    String username = "test";
    String password = "test123";

    try (Connection conn = DriverManager.getConnection(url, username,
password);
Scanner scanner = new Scanner(System.in)) {

        int choice;
        do {
            choice = scanner.nextInt();

            switch (choice) {
                case 1 -> addEmployee(conn, scanner);
                case 2 -> updateSalary(conn, scanner);
                case 3 -> viewEmployeeRecord(conn, scanner);
                case 4 -> displayAllEmployees(conn);
                case 5 -> System.out.println("Exiting Employee Management
System.");
                default -> System.out.println("Invalid choice. Please try again.");
            }
        } while (choice != 5);

    } catch (SQLException e) {
        System.out.println("Database Error: " + e.getMessage());
    }
}
```

## 2. Problem Statement

Create a JDBC-based Inventory Management System that handles runtime input to manage items in an inventory. The system should allow users to:

- Add a new item (item ID, name, quantity, price).
- Restock an item by increasing its quantity.
- Reduce the stock of an item, ensuring sufficient quantity.
- Display all items in the inventory in a sorted order by item ID.
- Exit the application.

Half of the code is given here; Only the remaining part should be completed.

The system should connect to a MySQL database using the following default credentials:

DB URL: jdbc:mysql://localhost/ri\_db

USER: test

PWD: test123

The items table has already been created with the following structure:

Table Name: items

### ***Input Format***

The first line of input consists of an integer choice, representing the operation to be performed (1 for Add Item, 2 for Restock item, 3 for reduce item, 4 for Display, 5 for Exit).

For choice 1 (Add Item):

- The second line consists of an integer item\_id.
- The third line consists of a string name.
- The fourth line consists of an integer quantity.
- The fifth line consists of a double price.

For choice 2 (Restock Item):

- The second line consists of an integer item\_id.
- The third line consists of an integer quantity\_to\_add (must be positive).

For choice 3 (Reduce Stock):

- The second line consists of an integer item\_id.
- The third line consists of an integer quantity\_to\_remove (must be positive).

For choice 4 (Display Inventory):

- No additional inputs are required.

For choice 5 (Exit):

- No additional inputs are required.

#### ***Output Format***

For choice 1 (Add Item):

- Print "Item added successfully" if the item was added.
- Print "Failed to add item." if the insertion failed.

For choice 2 (Restock Item):

- Print "Item restocked successfully" if the restock was successful.
- Print "Item not found." if the specified item ID does not exist.

For choice 3 (Reduce Stock):

- Print "Stock reduced successfully" if the stock reduction was successful.
- Print "Not enough stock to remove." if there is insufficient quantity.
- Print "Item not found." if the specified item ID does not exist.

For choice 4 (Display Inventory):

- Display each item on a new line in the format:
- ID | Name | Quantity | Price
- If no items are available, print nothing (or handle with an appropriate message if desired).

For choice 5 (Exit):

- Print "Exiting Inventory Management System."

For invalid input:

- Print "Invalid choice. Please try again."

### **Sample Test Case**

Input: 1

101

Laptop

50

1200.00

4

5

Output: Item added successfully

ID | Name | Quantity | Price

101 | Laptop | 50 | 1200.00

Exiting Inventory Management System.

### **Answer**

```
import java.sql.*;
import java.util.Scanner;

class InventoryManagementSystem {
    public static void main(String[] args) {
        try (Connection conn = DriverManager.getConnection("jdbc:mysql://
localhost/ri_db", "test", "test123");
        Scanner scanner = new Scanner(System.in)) {

            boolean running = true;
            while (running) {
```

```
int choice = scanner.nextInt();

switch (choice) {
    case 1:
        addlItem(conn, scanner);
        break;
    case 2:
        restockItem(conn, scanner);
        break;
    case 3:
        reduceStock(conn, scanner);
        break;
    case 4:
        displayInventory(conn);
        break;
    case 5:
        System.out.println("Exiting Inventory Management System.");
        running = false;
        break;
    default:
        System.out.println("Invalid choice. Please try again.");
    }
}

} catch (SQLException e) {
    e.printStackTrace();
}

}

public static void addlItem(Connection conn, Scanner scanner) {
    int itemId = scanner.nextInt();
    scanner.nextLine();

    String name = scanner.nextLine();

    int quantity = scanner.nextInt();

    double price = scanner.nextDouble();

    String insertQuery = "INSERT INTO items (item_id, name, quantity, price)
VALUES (?, ?, ?, ?)";
    try (PreparedStatement stmt = conn.prepareStatement(insertQuery)) {
        stmt.setInt(1, itemId);
        stmt.setString(2, name);
```

```
        stmt.setInt(3, quantity);
        stmt.setDouble(4, price);

        int rowsInserted = stmt.executeUpdate();
        System.out.println(rowsInserted > 0 ? "Item added successfully" : "Failed
to add item.");
    } catch (SQLException e) {
        System.out.println("Error adding item: " + e.getMessage());
    }
}
```

```
public static void restockItem(Connection conn, Scanner scanner) {
    int itemId = scanner.nextInt();
```

```
    int quantityToAdd = scanner.nextInt();
```

```
    // Check if the quantity is positive
```

```
    if (quantityToAdd <= 0) {
```

```
        System.out.println("Quantity to add must be positive.");
```

```
        return;
```

```
}
```

```
    String updateQuery = "UPDATE items SET quantity = quantity + ? WHERE
item_id = ?";
```

```
    try (PreparedStatement stmt = conn.prepareStatement(updateQuery)) {
```

```
        stmt.setInt(1, quantityToAdd);
```

```
        stmt.setInt(2, itemId);
```

```
        int rowsUpdated = stmt.executeUpdate();
```

```
        System.out.println(rowsUpdated > 0 ? "Item restocked successfully" :
```

```
"Item not found.");
```

```
    } catch (SQLException e) {
```

```
        System.out.println("Error during restock: " + e.getMessage());
```

```
}
```

```
}
```

```
public static void reduceStock(Connection conn, Scanner scanner) {
```

```
    int itemId = scanner.nextInt();
```

```
    int quantityToRemove = scanner.nextInt();
```

```
    // Check if the quantity is positive
```

```
if (quantityToRemove <= 0) {
    System.out.println("Quantity to remove must be positive.");
    return;
}

String checkQuantityQuery = "SELECT quantity FROM items WHERE item_id
= ?";
String updateQuery = "UPDATE items SET quantity = quantity - ? WHERE
item_id = ?";

try (PreparedStatement checkStmt =
conn.prepareStatement(checkQuantityQuery)) {
    checkStmt.setInt(1, itemId);
    ResultSet rs = checkStmt.executeQuery();

    if (rs.next()) {
        int currentQuantity = rs.getInt("quantity");

        if (currentQuantity >= quantityToRemove) {
            try (PreparedStatement stmt =
conn.prepareStatement(updateQuery)) {
                stmt.setInt(1, quantityToRemove);
                stmt.setInt(2, itemId);

                int rowsUpdated = stmt.executeUpdate();
                System.out.println(rowsUpdated > 0 ? "Stock reduced
successfully" : "Failed to reduce stock.");
            }
        } else {
            System.out.println("Not enough stock to remove.");
        }
    } else {
        System.out.println("Item not found.");
    }
} catch (SQLException e) {
    System.out.println("Error during stock reduction: " + e.getMessage());
}
}

public static void displayInventory(Connection conn) {
String displayQuery = "SELECT * FROM items ORDER BY item_id";
try (Statement stmt = conn.createStatement());
```

```
ResultSet rs = stmt.executeQuery(displayQuery)) {  
  
    System.out.println("ID | Name | Quantity | Price");  
    while (rs.next()) {  
        System.out.printf("%d | %s | %d | %.2f%n",  
            rs.getInt("item_id"),  
            rs.getString("name"),  
            rs.getInt("quantity"),  
            rs.getDouble("price"));  
    }  
} catch (SQLException e) {  
    System.out.println("Error displaying inventory: " + e.getMessage());  
}  
}  
}
```

**Status :** Correct

**Marks :** 10/10