Rajalakshmi Engineering College

Name: Dharshini A
Email: 241001049@rajalakshmi.edu.in
Roll no: 241001049
Phone: 8056618801
Branch: REC
Department: IT - Section 2
Batch: 2028
Degree: B.E - IT

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;

    public Vehicle(String regNumber, String ownerName, String vehicleType)

{


        this.regNumber = regNumber;
        this.ownerName = ownerName;
        this.vehicleType = vehicleType;


}

    @Override
    public boolean equals(Object obj)

{


        if (this == obj) return true;
        if (!(obj instanceof Vehicle)) return false;
        Vehicle other = (Vehicle) obj;
        return this.regNumber.equals(other.regNumber);


}

    @Override
    public int hashCode()

{


        return regNumber.hashCode();


}

    @Override
    public String toString()
```

```
{

      return regNumber + " " + ownerName + " " + vehicleType;

}

}
class TollBoothTracker

{

   public static void main(String[] args)

{

      Scanner sc = new Scanner(System.in);
      int n = Integer.parseInt(sc.nextLine());

      Set<Vehicle> vehicleSet = new HashSet<>();

      for (int i = 0; i < n; i++)

{

         String[] parts = sc.nextLine().split(" ");
         String regNumber = parts[0];
         String ownerName = parts[1];
         String vehicleType = parts[2];

         Vehicle vehicle = new Vehicle(regNumber, ownerName, vehicleType);
         vehicleSet.add(vehicle);

}

      for (Vehicle v : vehicleSet)
{

          System.out.println(v);

}

}

}
```

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

Rajalakshmi Engineering College

Name: Dharshini A
Email: 241001049@rajalakshmi.edu.in
Roll no: 241001049
Phone: 8056618801
Branch: REC
Department: IT - Section 2
Batch: 2028
Degree: B.E - IT

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.*;
import java.text.DecimalFormat;
class FruitFestival

{


  public static void main(String[] args)

{


    Scanner sc = new Scanner(System.in);
    Map<String, Double> fruitMap = new HashMap<>();
    DecimalFormat df = new DecimalFormat("0.00");

    while (true)

{


      String input = sc.nextLine();
```

```
        if (input.equalsIgnoreCase("done"))
    {

            break;

    }
        if (!input.contains(":") || input.indexOf(":") != input.lastIndexOf(":"))

    {

            System.out.println("Invalid format");
            return;

    }

        String[] parts = input.split(":");
        if (parts.length != 2)

    {

            System.out.println("Invalid format");
            return;

    }

        String fruitName = parts[0];
        String quantityStr = parts[1];

        try

    {

            double quantity = Double.parseDouble(quantityStr);
            fruitMap.put(fruitName, quantity);
```

```
} catch (NumberFormatException e)

{


            System.out.println("Invalid input");
            return;


}


}

    double total = 0.0;
    for (double qty : fruitMap.values())

{


      total += qty;


}

    System.out.println(df.format(total));


}

}
```

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

Rajalakshmi Engineering College

Name: Dharshini A
Email: 241001049@rajalakshmi.edu.in
Roll no: 241001049
Phone: 8056618801
Branch: REC
Department: IT - Section 2
Batch: 2028
Degree: B.E - IT

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 static void main(String[] args)

{
        Scanner sc = new Scanner(System.in);
        int n = Integer.parseInt(sc.nextLine());

        TreeMap<Character, Integer> frequencyMap = new TreeMap<>();

        for (int i = 0; i < n; i++)

    {


            String line = sc.nextLine();
            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());


    }
```

```
}
}
```

Status : Correct *Marks : 10/10*

Rajalakshmi Engineering College

Name: Dharshini A
Email: 241001049@rajalakshmi.edu.in
Roll no: 241001049
Phone: 8056618801
Branch: REC
Department: IT - Section 2
Batch: 2028
Degree: B.E - IT

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.*;

public class Main

{


   public static void main(String[] args)

{


      Scanner sc = new Scanner(System.in);
      int n = sc.nextInt();
      TreeSet<Integer> seats = new TreeSet<>();
      for (int i = 0; i < n; i++)

{


         seats.add(sc.nextInt());
}
    int m = sc.nextInt();
    if (seats.contains(m))
      System.out.println(m + " is present!");
    else
      System.out.println(m + " is not present!");


}

}
```

Status : Correct *Marks : 10/10*

Rajalakshmi Engineering College

Name: Dharshini A
Email: 241001049@rajalakshmi.edu.in
Roll no: 241001049
Phone: 8056618801
Branch: REC
Department: IT - Section 2
Batch: 2028
Degree: B.E - IT

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

```
// 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;


}

    @Override
    public int compareTo(Student other)

{


        if (Double.compare(this.gpa, other.gpa) != 0)

{


            return Double.compare(this.gpa, other.gpa);


} else

{


            return this.name.compareTo(other.name);


}


}

    @Override
    public String toString()
```

```
{

    return id + " " + name + " " + String.format("%.2f", gpa);

}

  @Override
  public boolean equals(Object o)

{

    if (this == o) return true;
    if (!(o instanceof Student)) return false;
    Student s = (Student) o;
    return this.id == s.id;

}

  @Override
  public int hashCode()

{

    return Objects.hash(id);

}

}
class StudentRecords

{

  public static void main(String[] args)

{
    Scanner sc = new Scanner(System.in);
    int n = Integer.parseInt(sc.nextLine());

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

    for (int i = 0; i < n; i++)

{


        String line = sc.nextLine();
        String[] parts = line.split(" ");
        int id = Integer.parseInt(parts[0]);
        String name = parts[1];
        double gpa = Double.parseDouble(parts[2]);

        students.add(new Student(id, name, gpa));


}

    for (Student s : students)

{


        System.out.println(s);


}


}

}
```

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

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

{


   public static void main(String[] args)

{


      Scanner sc = new Scanner(System.in);
      int n = Integer.parseInt(sc.nextLine());

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

      for (int i = 0; i < n; i++)

{


         String line = sc.nextLine();
         String[] parts = line.split(" ", 2);
         String time = parts[0];
         String description = parts[1];

         // Only add if time is not already present
         events.putIfAbsent(time, description);


}

      System.out.println("Scheduled Events:");
      for (Map.Entry<String, String> entry : events.entrySet())

{
        System.out.println(entry.getKey() + " - " + entry.getValue());

}


}

}
```

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.*;

public class Main

{


    public static void main(String[] args)

{


        Scanner sc = new Scanner(System.in);
        int N = sc.nextInt();
        String s = sc.next();
        HashMap<Character, Integer> map = new HashMap<>();
        for (char c : s.toCharArray())

{


            map.put(c, map.getOrDefault(c, 0) + 1);


}
        char result = '-';
        for (char c : s.toCharArray())

{


            if (map.get(c) == 1)

{


                result = c;
```

```
          break;

    }


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


    }

    }
```

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

Rajalakshmi Engineering College

Name: Dharshini A
Email: 241001049@rajalakshmi.edu.in
Roll no: 241001049
Phone: 8056618801
Branch: REC
Department: IT - Section 2
Batch: 2028
Degree: B.E - IT

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

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.*;

```
import java.util.*;

class Book

{


    private int isbn;
    private String title;
    private String author;

    public Book(int isbn, String title, String author)

{


        this.isbn = isbn;
        this.title = title;
        this.author = author;


}

    public int getIsbn()

{


        return isbn;


}

    public String toString()

{


        return "ISBN: " + isbn + ", Title: " + title + ", Author: " + author;


}
```

```
   public boolean equals(Object o)

{


      if (this == o) return true;
      if (!(o instanceof Book)) return false;
      Book b = (Book) o;
      return isbn == b.isbn;


}

   public int hashCode()

{


      return Objects.hash(isbn);


}

}

class Library

{


   private LinkedHashSet<Book> books = new LinkedHashSet<>();

   public void addBook(int isbn, String title, String author)

{


      books.add(new Book(isbn, title, author));


}
```

```
    public void removeBook(int isbn)
  {

        books.removeIf(b -> b.getIsbn() == isbn);

  }
    public void displayBooks()
  {

        if (books.isEmpty())
  {

            System.out.println("No books available");

  } else
  {

            for (Book b : books)
  {

                System.out.println(b);

  }

  }
```

```
}

}

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***

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 times1 appears once3 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;


import java.util.HashMap;

class MajorityElementFinder

{


   public static int findMajorityElement(int[] arr)

{


      HashMap<Integer, Integer> voteCount = new HashMap<>();
      int n = arr.length;
      int threshold = n / 2;

      for (int vote : arr)

{


         voteCount.put(vote, voteCount.getOrDefault(vote, 0) + 1);
         if (voteCount.get(vote) > threshold)

{


            return vote;


}


}

      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

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;


}

    @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 == null || getClass() != obj.getClass()) return false;
        Student other = (Student) obj;
        return rollNo == other.rollNo;


}
```

```
    @Override
    public int hashCode()
{

        return Objects.hash(rollNo);

}

}

public class Main

{

    public static void main(String[] args)

{

        Scanner sc = new Scanner(System.in);
        int N = sc.nextInt();
        TreeSet<Student> students = new TreeSet<>();

        for (int i = 0; i < N; i++)
{

            String command = sc.next();
            if (command.equals("A"))

{

                int rollNo = sc.nextInt();
                String name = sc.next();
                students.add(new Student(rollNo, name));
```

```
} else if (command.equals("M"))

{


        int rollNo = sc.nextInt();
        // Find student and increase attendance
        for (Student s : students)

{


          if (s.rollNo == rollNo)

{


            s.attendance++;
            break;


}


}


} else if (command.equals("D"))

{


        for (Student s : students)

{


          System.out.println(s.rollNo + " " + s.name + " " + s.attendance);


}
```

```
}

}
        sc.close();

}

}
```

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.*;

import java.util.HashMap;
class ScoreTracker

{


   HashMap<String, Integer> scoreMap = new HashMap<>();
   boolean invalidInput = false;
   boolean invalidFormat = false;

   public boolean processInput(String input)

{


      // Check if exactly one ':' exists
      if (input.chars().filter(ch -> ch == ':').count() != 1)

{


         invalidFormat = true;
         System.out.println("Invalid format");
         return false;


}
      // Remove ':' for special char check
      String temp = input.replace(":", "");
```

```
        if (!temp.matches("[a-zA-Z0-9]+"))
    {

           invalidFormat = true;
           System.out.println("Invalid format");
           return false;

    }
        String[] parts = input.split(":");
        if (parts.length != 2)
    {

           invalidFormat = true;
           System.out.println("Invalid format");
           return false;

    }
        String player = parts[0].trim();
        String scoreStr = parts[1].trim();

        if (player.length() < 1 || player.length() > 20)
    {

           invalidInput = true;
           System.out.println("Invalid input");
           return false;

    }
        int score = 0;
        try
    {
         score = Integer.parseInt(scoreStr);

   } catch (NumberFormatException e)

   {


         invalidInput = true;
         System.out.println("Invalid input");
         return false;


   }
       if (score < 1 || score > 100)

   {


         invalidInput = true;
         System.out.println("Invalid input");
         return false;


   }
       scoreMap.put(player, score);
       return true;


   }

     public String findTopPlayer()

   {


       String topPlayer = "";
       int maxScore = Integer.MIN_VALUE;
       for (String name : scoreMap.keySet())

   {
```

```
            int val = scoreMap.get(name);
            if (val > maxScore)

    {



                maxScore = val;
                topPlayer = name;



    }



    }
        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***

Rajalakshmi Engineering College

Name: Dharshini A
Email: 241001049@rajalakshmi.edu.in
Roll no: 241001049
Phone: 8056618801
Branch: REC
Department: IT - Section 2
Batch: 2028
Degree: B.E - IT

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 : 10

Section 1 : Project

1. Problem Statement

Create a JDBC-based School Management System that handles runtime input to manage student records. The system should allow users to:

Add a new student (student ID, name, grade level, GPA).

Update a student's GPA, ensuring the GPA value is within the valid range (0.0 - 4.0).

View a specific student’s record by student ID.

Display all students in the database.

Exit the application.

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 students table has already been created with the following structure:

Table Name: students

Input Format

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

(1 for Add Student, 2 for Update GPA, 3 for View Student Record, 4 for Display All Students, 5 for Exit)

For choice 1 (Add Student):

- The second line consists of an integer student_id.
- The third line consists of a string name.
- The fourth line consists of a string grade_level.
- The fifth line consists of a double gpa (must be between 0.0 and 4.0).

For choice 2 (Update GPA):

- The second line consists of an integer student_id.
- The third line consists of a double new_gpa (must be between 0.0 and 4.0).

For choice 3 (View Student Record):

- The second line consists of an integer student_id.

For choice 4 (Display All Students):

- No additional inputs are required.

For choice 5 (Exit):

- No additional inputs are required.

Output Format

The output displays:

For choice 1 (Add Student):

- Print "Student added successfully" if the student was added.
- Print "Failed to add student." if the insertion failed.

For choice 2 (Update GPA):

- Print "GPA updated successfully" if the GPA update was successful.
- Print "Student not found." if the specified student ID does not exist.
- Print "GPA must be between 0.0 and 4.0." if the provided GPA is out of the valid range.

For choice 3 (View Student Record):

- Display the student details in the format:
- ID: [student_id] | Name: [name] | Grade Level: [grade_level] | GPA: [gpa]
- Print "Student not found." if the specified student ID does not exist.

For choice 4 (Display All Students):

- Display each student on a new line in the format:
- ID | Name | Grade Level | GPA
- If there are no records, print nothing (or handle with an appropriate message if desired).

For choice 5 (Exit):

- Print "Exiting School Management System."

For invalid input:

- Print "Invalid choice. Please try again."

Sample Test Case

Input: 1
101
Alice Johnson
10

```
3.8
5
Output: Student added successfully
Exiting School Management System.
```

Answer

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

class SchoolManagementSystem {
   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:
                  addStudent(conn, scanner);
                  break;
               case 2:
                  updateGrades(conn, scanner);
                  break;
               case 3:
                  viewStudentRecord(conn, scanner);
                  break;
               case 4:
                  displayAllStudents(conn);
                  break;
               case 5:
                  System.out.println("Exiting School Management System.");
                  running = false;
                  break;
               default:
                  System.out.println("Invalid choice. Please try again.");
            }
         }
```

```
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    public static void addStudent(Connection conn, Scanner scanner){



        int studentId = scanner.nextInt();
        scanner.nextLine(); // consume newline
        String name = scanner.nextLine();
        String gradeLevel = scanner.nextLine();
        double gpa = scanner.nextDouble();
        scanner.nextLine(); // consume newline

        if (gpa < 0.0 || gpa > 4.0)

   {


            System.out.println("GPA must be between 0.0 and 4.0.");
            return;


   }
        String sql = "INSERT INTO students (student_id, name, grade_level, gpa)
   VALUES (?, ?, ?, ?)";
        try (PreparedStatement pstmt = conn.prepareStatement(sql))

   {


            pstmt.setInt(1, studentId);
            pstmt.setString(2, name);
            pstmt.setString(3, gradeLevel);
            pstmt.setDouble(4, gpa);
            int rows = pstmt.executeUpdate();
            if (rows > 0)

   {
```

```
            System.out.println("Student added successfully");

    } else

    {


            System.out.println("Failed to add student.");


    }


    } catch (SQLException e)

    {


           System.out.println("Failed to add student.");


    }


    }
      public static void updateGrades(Connection conn, Scanner scanner)

    {


        int studentId = scanner.nextInt();
        double newGpa = scanner.nextDouble();
        scanner.nextLine(); // consume newline

        if (newGpa < 0.0 || newGpa > 4.0)

    {
            System.out.println("GPA must be between 0.0 and 4.0.");
            return;

}
        try (PreparedStatement pstmt = conn.prepareStatement("UPDATE students
SET gpa = ? WHERE student_id = ?"))

{


            pstmt.setDouble(1, newGpa);
            pstmt.setInt(2, studentId);
            int rows = pstmt.executeUpdate();
            if (rows > 0)

{


                System.out.println("GPA updated successfully");


} else

{


                System.out.println("Student not found.");


}


} catch (SQLException e)

{


            System.out.println("Student not found.");


}
```

```
}
    public static void viewStudentRecord(Connection conn, Scanner scanner)

{


        int studentId = scanner.nextInt();
        scanner.nextLine(); // consume newline
        try (PreparedStatement pstmt = conn.prepareStatement("SELECT * FROM
students WHERE student_id = ?"))

{


            pstmt.setInt(1, studentId);
            ResultSet rs = pstmt.executeQuery();
            if (rs.next())

{


                System.out.printf("ID: %d | Name: %s | Grade Level: %s | GPA: %.2f%n",
                        rs.getInt("student_id"),
                        rs.getString("name"),
                        rs.getString("grade_level"),
                        rs.getDouble("gpa"));


} else

{


                System.out.println("Student not found.");


}
```

```
} catch (SQLException e)
{

        System.out.println("Student not found.");

}

}
   public static void displayAllStudents(Connection conn)
{

      try (Statement stmt = conn.createStatement())
{

         ResultSet rs = stmt.executeQuery("SELECT * FROM students");
         boolean headerPrinted = false;
         while (rs.next())
{

            if (!headerPrinted)
{

               System.out.println("ID | Name | Grade Level | GPA");
               headerPrinted = true;

}
            System.out.printf("%d | %s | %s | %.2f%n",
                  rs.getInt("student_id"),
                    rs.getString("name"),
                    rs.getString("grade_level"),
                    rs.getDouble("gpa"));

}

} catch (SQLException e)

{

     // Silent fail or print nothing

}
}

}
```

Status : Wrong ***Marks : 0/10***

2. 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_dbUsername: testPassword: 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());
        }
    }
}
```

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