Rajalakshmi Engineering College

Name: Pratiba S
Email: 241901080@rajalakshmi.edu.in
Roll no: 241901080
Phone: 7200010194
Branch: REC
Department: CSE (CS) - Section 2
Batch: 2028
Degree: B.E - CSE (CS)

Scan to verify results

2024_28_III_OOPS Using Java Lab

2028_REC_OOPS using Java_Week 5_MCQ

Attempt : 1
Total Mark : 15
Marks Obtained : 14

Section 1 : MCQ

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

```
class A {
   int y = 30;
}

public class Main {
   public static void main(String[] args) {
      A a1 = new A();
      A a2 = new A();
      a1.y = 50;
      System.out.println(a2.y);
   }
}
```

Answer

30

Status : Correct **Marks : 1/1**

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

```
class A {
   int val = 20;
}

public class Main {
   public static void main(String[] args) {
      A obj1 = new A();
      A obj2 = obj1;
      obj2.val += 5;
      System.out.println(obj1.val);
   }
}
```

Answer

5

Status : Wrong **Marks : 0/1**

3. What is the output of the following code?

```
class Box {
   int height;
   Box(int height) {
      this.height = height;
   }
   void modifyHeight(Box b) {
      b.height += 10;
   }
}
public class Main {
   public static void main(String[] args) {
      Box b1 = new Box(20);
      b1.modifyHeight(b1);
```

```
        System.out.println(b1.height);
    }
}
```

Answer

30

Status : Correct ***Marks : 1/1***

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

```
class Test {
    private int value;
    Test(int value) {
        this.value = value;
    }
    public int getValue() {
        return value;
    }
}
public class Main {
    public static void main(String[] args) {
        Test obj = new Test(10);
        System.out.println(obj.value);
    }
}
```

Answer

Compile-time error

Status : Correct ***Marks : 1/1***

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

```
class MathUtils {
    int add(int x) {
        return x + x;
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        MathUtils m = new MathUtils();
        System.out.println(m.add(5));
    }
}
```

Answer

10

Status : Correct ***Marks : 1/1***

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

```
class A {
    int p = 5;
    int q = 2;
}

class Main {
    public static void main(String[] args) {
        A obj = new A();
        System.out.println(obj.p + obj.q);
    }
}
```

Answer

7

Status : Correct ***Marks : 1/1***

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

```
class Demo {
    void printMessage() {
        System.out.println("Hello from Demo");
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        Demo d = new Demo();
        d.printMessage();
    }
}
```

Answer

Hello from Demo

Status : Correct ***Marks : 1/1***

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

```
class Person {
    String name;
    void setName(String n) {
        name = n;
    }
    void printName() {
        System.out.println(name);
    }
}

class Test {
    public static void main(String[] args) {
        Person p = new Person();
        p.printName();
    }
}
```

Answer

null

Status : Correct ***Marks : 1/1***

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

```
class Person {
    int age = 18;
}

public class Main {
    public static void main(String[] args) {
        Person p = new Person();
        p.age += 2;
        System.out.println("Age: " + p.age);
    }
}
```

Answer

Age: 20

Status : Correct ***Marks : 1/1***

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

```
class Box {
    int volume(int l, int b, int h) {
        return l * b * h;
    }
}

public class Main {
    public static void main(String[] args) {
        Box b = new Box();
        System.out.println(b.volume(2, 3, 4));
    }
}
```

Answer

24

Status : Correct ***Marks : 1/1***

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

```
class Box {
    int length = 5;
    int width = 4;

    int area() {
        return length * width;
    }

    public static void main(String[] args) {
        Box b = new Box();
        System.out.println("Area = " + b.area());
    }
}
```

Answer

Area = 20

Status : Correct ***Marks : 1/1***

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

```
class A {
    int x = 50;
}

public class Main {
    public static void main(String[] args) {
        A obj1 = new A();
        A obj2 = obj1;
        obj2.x = 100;
        System.out.println(obj1.x);
    }
}
```

Answer

100

Status : Correct ***Marks : 1/1***

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

```
class Ball {
   int size = 11;
}

class Game {
   public static void main(String[] args) {
      Ball b1 = new Ball();
      Ball b2 = new Ball();
      b2.size = 10;
      System.out.println(b1.size);
   }
}
```

Answer

11

Status : Correct ***Marks : 1/1***

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

```
class Alpha {
   void greet(String name) {
      System.out.println("Hello " + name);
   }
}

public class Main {
   public static void main(String[] args) {
      Alpha obj = new Alpha();
      obj.greet("Anu");
   }
}
```

Answer

Hello Anu

Status : Correct ***Marks : 1/1***

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

```
class Sample {
    int x = 10;

    void display() {
        System.out.println("x = " + x);
    }

    public static void main(String[] args) {
        Sample s = new Sample();
        s.display();
    }
}
```

Answer

x = 10

Status : Correct ***Marks : 1/1***

Rajalakshmi Engineering College

Name: Pratiba S
Email: 241901080@rajalakshmi.edu.in
Roll no: 241901080
Phone: 7200010194
Branch: REC
Department: CSE (CS) - Section 2
Batch: 2028
Degree: B.E - CSE (CS)

Scan to verify results

2024_28_III_OOPS Using Java Lab

2028_REC_OOPS using Java_Week 5_Q2

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

You are working as a developer for CityBank, which wants to build a basic account management system.

Each customer at the bank has:

An Account Number (integer)A Customer Name (string)An Initial Balance (double)

The bank allows two types of transactions:

Deposit – increases the balance.Withdrawal – decreases the balance only if enough funds are available.

If the withdrawal amount is greater than the balance, the withdrawal should not happen, and the balance should remain the same.

You are required to implement this system using:

A class with attributes for account details.A constructor to initialize account details.Setter methods to update details if needed.Getter methods to retrieve details.Objects of the class to represent customers.

Finally, display each customer's account details after all transactions.

Input Format

The first line of input contains an integer N, representing the number of customers.

For each customer:

- The next line contains the account number (integer).
- The following line contains the customer name (string).
- The next line contains the initial balance (double).
- The next line contains the deposit amount (double).
- The next line contains the withdrawal amount (double).

Output Format

For each customer, print the details in the following format:

1. Account Number: <account_number>
2. Customer Name: <customer_name>
3. Final Balance: <final_balance> (rounded to one decimal place)

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 1
1234
Rahul Sharma
5000
2000
3000
Output: Account Number: 1234
Customer Name: Rahul Sharma

Final Balance: 4000.0

Answer

```
import java.util.*;
class Account {
    int accountNumber;
    String customerName;
    double balance;
    Account(int accountNumber, String customerName, double balance) {
        this.accountNumber = accountNumber;
        this.customerName = customerName;
        this.balance = balance;
    }
    void deposit(double amount) {
        if (amount > 0) {
            balance += amount;
        }
    }
    void withdraw(double amount) {
        if (amount > 0 && amount <= balance) {
            balance -= amount;
        }
    }
    int getAccountNumber() {
        return accountNumber;
    }
    String getCustomerName() {
        return customerName;
    }
    double getBalance() {
        return balance;
    }
    void setCustomerName(String customerName) {
        this.customerName = customerName;
    }
    void setBalance(double balance) {
        this.balance = balance;
    }
}

class Main {
    public static void main(String[] args) {
```

```
        Scanner sc = new Scanner(System.in);
        int N = Integer.parseInt(sc.nextLine());
        for (int i = 0; i < N; i++) {
            int accNo = Integer.parseInt(sc.nextLine());
            String name = sc.nextLine();
            double initBalance = Double.parseDouble(sc.nextLine());
            double depositAmt = Double.parseDouble(sc.nextLine());
            double withdrawAmt = Double.parseDouble(sc.nextLine());
            Account acc = new Account(accNo, name, initBalance);
            acc.deposit(depositAmt);
            acc.withdraw(withdrawAmt);
            System.out.printf("Account Number: %d%n", acc.getAccountNumber());
            System.out.printf("Customer Name: %s%n", acc.getCustomerName());
            System.out.printf("Final Balance: %.1f%n", acc.getBalance());
        }
        sc.close();
    }
}
```

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

Rajalakshmi Engineering College

Name: Pratiba S
Email: 241901080@rajalakshmi.edu.in
Roll no: 241901080
Phone: 7200010194
Branch: REC
Department: CSE (CS) - Section 2
Batch: 2028
Degree: B.E - CSE (CS)

Scan to verify results

2024_28_III_OOPS Using Java Lab

2028_REC_OOPS using Java_Week 5_Q3

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

Neha is working as a developer for CityElectricity Board, which wants to build a household electricity billing system.

Each customer's electricity account has:

A Customer ID (integer)A Customer Name (string)Units Consumed (double)

The electricity bill is calculated based on these rules:

For the first 100 units 5 units charge per unitFor the next 100 units (101–200) 7 units charge per unitFor units above 200 10 units charge per unitIf the total bill exceeds 2000 units, a 5% discount is applied on the final bill.

Neha has been asked to implement this system using:

A class with attributes for customer details.A constructor to initialize customer details.Setter methods to update details if needed.Getter methods to retrieve details.Objects of the class to represent customers.

Finally, display each customer's details and final bill amount.

Input Format

The first line of input contains an integer N, representing the number of customers.

For each customer:

- The next line contains the Customer ID (integer).
- The following line contains the Customer Name (string).
- The next line contains the Units Consumed (double).

Output Format

For each customer, print the details in the following format:

Customer ID: <customer_id>

Customer Name: <customer_name>

Final Bill: <final_bill> (rounded to one decimal place)

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 1
1001
Ravi Kumar
80
Output: Customer ID: 1001
Customer Name: Ravi Kumar
Final Bill: 400.0

Answer

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

```
class Customer {
    int customerId;
    String customerName;
    double unitsConsumed;
    Customer(int customerId, String customerName, double unitsConsumed) {
        this.customerId = customerId;
        this.customerName = customerName;
        this.unitsConsumed = unitsConsumed;
    }
    double calculateBill() {
        double bill;
        if (unitsConsumed <= 100) {
            bill = unitsConsumed * 5;
        } else if (unitsConsumed <= 200) {
            bill = (100 * 5) + (unitsConsumed - 100) * 7;
        } else {
            bill = (100 * 5) + (100 * 7) + (unitsConsumed - 200) * 10;
        }
        if (bill > 2000) {
            bill = bill * 0.95;
        }
        return bill;
    }
    int getCustomerId() {
        return customerId;
    }
    String getCustomerName() {
        return customerName;
    }
    double getUnitsConsumed() {
        return unitsConsumed;
    }
    void setCustomerName(String name) {
        this.customerName = name;
    }
    void setUnitsConsumed(double units) {
        this.unitsConsumed = units;
    }
}

class Main {
    public static void main(String[] args) {
```

```
        Scanner sc = new Scanner(System.in);
        int N = Integer.parseInt(sc.nextLine());
        for (int i = 0; i < N; i++) {
            int id = Integer.parseInt(sc.nextLine());
            String name = sc.nextLine();
            double units = Double.parseDouble(sc.nextLine());
            Customer c = new Customer(id, name, units);
            System.out.printf("Customer ID: %d%n", c.getCustomerId());
            System.out.printf("Customer Name: %s%n", c.getCustomerName());
            System.out.printf("Final Bill: %.1f%n", c.calculateBill());
        }
        sc.close();
    }
}
```

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

Rajalakshmi Engineering College

Name: Pratiba S
Email: 241901080@rajalakshmi.edu.in
Roll no: 241901080
Phone: 7200010194
Branch: REC
Department: CSE (CS) - Section 2
Batch: 2028
Degree: B.E - CSE (CS)

Scan to verify results

2024_28_III_OOPS Using Java Lab

2028_REC_OOPS using Java_Week 5_Q4

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

You are working as a developer for CityCab, a taxi service company that wants to build a ride fare management system.

Each customer booking has:

A Booking ID (integer)A Customer Name (string)A Distance Travelled in km (double)

The fare calculation rules are:

Base Fare = 50 units (flat charge for every ride).Per km charge = 10 units/km.If the distance is greater than 20 km, a 10% discount is applied on the total fare.

You are required to implement this system using:

A class with attributes for booking details.A constructor to initialize booking details.Setter methods to update details if needed.Getter methods to retrieve details.Objects of the class to represent customer rides.

Finally, display each booking's details and final fare.

Input Format

The first line of input contains an integer N, representing the number of bookings.

For each booking:

- The next line contains the booking ID (integer).
- The following line contains the customer's name (string).
- The next line contains the distance travelled (double).

Output Format

For each booking, print the details in the following format:

1. Booking ID: <booking_id>
2. Customer Name: <customer_name>
3. Final Fare: <final_fare> (rounded to one decimal place)

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 1
1234
Rahul Sharma
15

Output: Booking ID: 1234
Customer Name: Rahul Sharma
Final Fare: 200.0

Answer

```
import java.util.Scanner;
class Booking {
    private int bookingId;
```

```
    private String customerName;
    private double distance;
    public Booking(int bookingId, String customerName, double distance) {
        this.bookingId = bookingId;
        this.customerName = customerName;
        this.distance = distance;
    }
    public void setBookingId(int bookingId) {
        this.bookingId = bookingId;
    }
    public void setCustomerName(String customerName) {
        this.customerName = customerName;
    }
    public void setDistance(double distance) {
        this.distance = distance;
    }
    public int getBookingId() {
        return bookingId;
    }
    public String getCustomerName() {
        return customerName;
    }
    public double getDistance() {
        return distance;
    }
    public double calculateFare() {
        double baseFare = 50;
        double fare = baseFare + (distance * 10);
        if (distance > 20) {
            fare = fare - (fare * 0.10);
        }
        return fare;
    }
}
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = Integer.parseInt(sc.nextLine().trim());
        for (int i = 0; i < n; i++) {
            int bookingId = Integer.parseInt(sc.nextLine().trim());
            String customerName = sc.nextLine().trim();
            double distance = Double.parseDouble(sc.nextLine().trim());
```

```
            Booking booking = new Booking(bookingId, customerName, distance);
            System.out.println("Booking ID: " + booking.getBookingId());
            System.out.println("Customer Name: " + booking.getCustomerName());
            System.out.printf("Final Fare: %.1f%n", booking.calculateFare());
        }
        sc.close();
    }
}
```

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

Rajalakshmi Engineering College

Name: Pratiba S
Email: 241901080@rajalakshmi.edu.in
Roll no: 241901080
Phone: 7200010194
Branch: REC
Department: CSE (CS) - Section 2
Batch: 2028
Degree: B.E - CSE (CS)

Scan to verify results

2024_28_III_OOPS Using Java Lab

2028_REC_OOPS using Java_Week 5_PAH

Attempt : 2
Total Mark : 50
Marks Obtained : 49

Section 1 : Coding

1. Problem Statement

Neha is working as a developer for CityQuiz Platform, which wants to build a system to calculate quiz scores and identify top scorers among participants.

Each participant's record has:

Participant ID (integer)Participant Name (string)An array of scores in 5 quiz rounds (integers, each between 0 and 100)

The system must calculate:

Total Score = sum of scores in all 5 rounds.Average Score = Total Score ÷ 5.If a participant scores above 80 in all rounds, a bonus of 10 points is added to the total score.Identify the Top Scorer among all participants. If

two participants have the same total score, the one with the lower Participant ID is considered the top scorer.

Neha has been asked to implement this system using:

A class with attributes for participant details.A constructor to initialize participant details.Getter and setter methods to retrieve or update participant details.A method to calculate total score and average score (including bonus if applicable).Objects of the class to represent participants.

Finally, display each participant's details and announce the Top Scorer.

Input Format

The first line of input contains an integer N, representing the number of participants.

For each participant:

- Next line: Participant ID (integer)
- Next line: Participant Name (string)
- Next line: 5 integers separated by spaces (scores for 5 quiz rounds)

Output Format

For each participant:

- Participant ID: <participant_id>
- Participant Name: <participant_name>
- Total Score: <total_score>
- Average Score: <average_score>

Finally, print "Top Scorer: <participant_name> with <total_score> points"

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 1
1001
Ravi Kumar
85 90 88 92 87

Output: Participant ID: 1001
Participant Name: Ravi Kumar
Total Score: 452
Average Score: 90
Top Scorer: Ravi Kumar with 452 points

Answer

```
import java.util.Scanner;
class Participant {
   private int participantId;
   private String participantName;
   private int[] scores;
   public Participant(int participantId, String participantName, int[] scores) {
      this.participantId = participantId;
      this.participantName = participantName;
      this.scores = scores;
   }
   public int getParticipantId() {
      return participantId;
   }
   public String getParticipantName() {
      return participantName;
   }
   public int[] getScores() {

      return scores;
   }
   public void setParticipantId(int participantId) {
      this.participantId = participantId;
   }
   public void setParticipantName(String participantName) {
      this.participantName = participantName;
   }

   public void setScores(int[] scores) {
      this.scores = scores;
   }
   public int calculateTotalScore() {
```

```
        int total = 0;
        boolean allAbove80 = true;
        for (int score : scores) {
            total += score;
            if (score <= 80) {
                allAbove80 = false;
            }
        }
        if (allAbove80) {
            total += 10;
        }
        return total;
    }
    public double calculateAverageScore() {
        return calculateTotalScore() / 5.0;
    }
}
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        if (!sc.hasNextInt()) {
            System.err.println("No input available for number of participants.");
            sc.close();
            return;
        }
        int N = sc.nextInt();
        sc.nextLine();
        if (N < 1 || N > 50) {
            System.err.println("Invalid number of participants: " + N);
            sc.close();
            return;
        }
        Participant[] participants = new Participant[N];
        for (int i = 0; i < N; i++) {
            if (!sc.hasNextInt()) {
                System.err.println("Insufficient input for participant ID at participant " +
(i + 1));
                sc.close();
                return;
            }
            int participantId = sc.nextInt();
            if (participantId < 1000 || participantId > 9999) {
```

```
            System.err.println("Invalid participant ID at participant " + (i + 1));
            sc.close();
            return;
        }
        sc.nextLine();
        if (!sc.hasNextLine()) {
            System.err.println("Insufficient input for participant name at participant
" + (i + 1));
            sc.close();
            return;
        }
        String participantName = sc.nextLine().trim();
        if (participantName.isEmpty()) {
            System.err.println("Participant name cannot be empty at participant " +
(i + 1));
            sc.close();
            return;
        }
        int[] scores = new int[5];
        for (int j = 0; j < 5; j++) {
            if (!sc.hasNextInt()) {
                System.err.println("Insufficient input for scores at participant " + (i +
1));
                sc.close();
                return;


}
            scores[j] = sc.nextInt();
            if (scores[j] < 0 || scores[j] > 100)

{


                System.err.println("Invalid score at participant " + (i + 1));
                sc.close();
                return;


}
```

```
}
        // Consume newline only if more participants are expected
        if (i < N - 1)
{
            if (sc.hasNextLine())
{
                sc.nextLine();
} else
{
                System.err.println("Incomplete input for participant " + (i + 1));
                sc.close();
                return;
}
}
        participants[i] = new Participant(participantId, participantName, scores);
}
    // Print participant details
    for (Participant participant : participants)
{
```

```
        System.out.println("Participant ID: " + participant.getParticipantId());
        System.out.println("Participant Name: " +
participant.getParticipantName());
        System.out.println("Total Score: " + participant.calculateTotalScore());
        System.out.printf("Average Score: %.0f\n",
participant.calculateAverageScore());


}

    // Find top scorer
    Participant topScorer = participants[0];
    int maxScore = topScorer.calculateTotalScore();
    for (Participant participant : participants)

{


        int currentScore = participant.calculateTotalScore();
        if (currentScore > maxScore ||
            (currentScore == maxScore && participant.getParticipantId() <
topScorer.getParticipantId()))

{


            topScorer = participant;
            maxScore = currentScore;


}


}

    // Print top scorer
    System.out.println("Top Scorer: " + topScorer.getParticipantName() + " with "
+ maxScore + " points");

    sc.close();
```

}
}

Status : Partially correct ***Marks : 9/10***

2. Problem Statement

Anjali is working as a developer for CityFitness Gym, which wants to build a system to calculate monthly membership fees for gym members based on the type of membership and the number of personal training sessions booked.

Each member’s record has:

Member ID (integer)Member Name (string)Membership Type (string: "Basic", "Premium", "Elite")Number of Personal Training Sessions (integer)

The monthly fees are:

Basic – 1000 unitsPremium – 1500 unitsElite – 2000 units

The cost of personal training sessions is 500 units per session.

The calculation rules:

Total Amount = Membership Fee + (Number of Personal Training Sessions × 500)If the number of sessions is more than 5, a 10% discount is applied on the total amount.If the member has Elite membership and the total amount exceeds 4000, an additional 5% service tax is added after discount.

Anjali has been asked to implement this system using:

A class with attributes for member details.A constructor to initialize member details.Getter and Setter methods to retrieve and update member details if required.A method to calculate the final monthly fee.Objects of the class to represent members.

Finally, display each member’s details and the final monthly fee.

Input Format

The first line contains an integer N, representing the number of members.

For each member:

- Next line contains Member ID (integer)
- Next line contains Member Name (string)
- Next line contains Membership Type ("Basic", "Premium", "Elite")
- Next line contains Number of Personal Training Sessions (integer)

Output Format

For each member, print:

- Member ID: <member_id>
- Member Name: <member_name>
- Final Monthly Fee: <final_fee> (The final fee must be rounded to one decimal place)

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 1
1001
Ravi Kumar
Basic
3
Output: Member ID: 1001
Member Name: Ravi Kumar
Final Monthly Fee: 2500.0

Answer

```
import java.util.Scanner;
class Member {
    private int memberId;
    private String memberName;
    private String membershipType;
    private int trainingSessions;
    public Member(int memberId, String memberName, String membershipType,
int trainingSessions) {
        this.memberId = memberId;
```

```
        this.memberName = memberName;
        this.membershipType = membershipType;
        this.trainingSessions = trainingSessions;
    }
    public int getMemberId() {
        return memberId;
    }
    public String getMemberName() {
        return memberName;
    }
    public String getMembershipType() {
        return membershipType;
    }
    public int getTrainingSessions() {
        return trainingSessions;
    }
    public void setMemberId(int memberId) {
        this.memberId = memberId;
    }
    public void setMemberName(String memberName) {
        this.memberName = memberName;
    }
    public void setMembershipType(String membershipType) {
        this.membershipType = membershipType;
    }
    public void setTrainingSessions(int trainingSessions) {
        this.trainingSessions = trainingSessions;
    }
    public double calculateFinalFee() {
        double membershipFee;
        switch (membershipType) {
            case "Basic":
                membershipFee = 1000.0;
                break;
            case "Premium":
                membershipFee = 1500.0;
                break;
            case "Elite":
                membershipFee = 2000.0;
                break;
            default:
                membershipFee = 0.0;
```

```
        }
        double totalAmount = membershipFee + (trainingSessions * 500.0);
        if (trainingSessions > 5) {
            totalAmount *= 0.9;
        }
        if (membershipType.equals("Elite") && totalAmount > 4000) {
            totalAmount *= 1.05;
        }
        return totalAmount;
    }

}

public class Main
{

    public static void main(String[] args)

{


        Scanner sc = new Scanner(System.in);

        // Read number of members
        if (!sc.hasNextInt())

{


            System.err.println("No input available for number of members.");
            sc.close();
            return;


}
        int N = sc.nextInt();
        sc.nextLine(); // Consume newline

        // Validate N
        if (N < 1 || N > 100)
```

```
{
            System.err.println("Invalid number of members: " + N);
            sc.close();
            return;


}

        // Process each member
        for (int i = 0; i < N; i++)
{

            // Read and validate memberId
            if (!sc.hasNextInt())

{


                System.err.println("Insufficient input for member ID at member " + (i +
1));
                sc.close();
                return;


}
            int memberId = sc.nextInt();
            if (memberId < 1000 || memberId > 9999)

{


                System.err.println("Invalid member ID at member " + (i + 1));
                sc.close();
                return;


}
```

```
            sc.nextLine(); // Consume newline

            // Read and validate memberName
            if (!sc.hasNextLine())

{


                System.err.println("Insufficient input for member name at member " + (i
+ 1));
                sc.close();
                return;


}
            String memberName = sc.nextLine().trim();
            if (memberName.isEmpty())

{


                System.err.println("Member name cannot be empty at member " + (i +
1));
                sc.close();
                return;


}

            // Read and validate membershipType
            if (!sc.hasNextLine())

{


                System.err.println("Insufficient input for membership type at member "
+ (i + 1));
                sc.close();
                return;


}
```

```
        String membershipType = sc.nextLine().trim();
        if (!membershipType.equals("Basic") && !
membershipType.equals("Premium") && !membershipType.equals("Elite"))

{


            System.err.println("Invalid membership type at member " + (i + 1));
            sc.close();
            return;


}

        // Read and validate trainingSessions
        if (!sc.hasNextInt())

{


            System.err.println("Insufficient input for training sessions at member " +
(i + 1));
            sc.close();
            return;


}
        int trainingSessions = sc.nextInt();
        if (trainingSessions < 0)

{


            System.err.println("Invalid number of training sessions at member " + (i
+ 1));
            sc.close();
            return;


}

        // Consume newline only if more members are expected
```

```
        if (i < N - 1)
{

            if (sc.hasNextLine())

{


                sc.nextLine();


} else
{

                System.err.println("Incomplete input for member " + (i + 1));
                sc.close();
                return;


}


}
        // Create Member object
        Member member = new Member(memberId, memberName,
membershipType, trainingSessions);

        // Calculate final fee
        double finalFee = member.calculateFinalFee();

        // Print output
        System.out.println("Member ID: " + member.getMemberId());
        System.out.println("Member Name: " + member.getMemberName());
        System.out.printf("Final Monthly Fee: %.1f\n", finalFee);


}
```

```
        sc.close();

    }

    }
```

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

3. Problem Statement

Neha is working as a developer for CityMovie Theatre, which wants to build a system to calculate total ticket cost for movie-goers based on the number of tickets and type of seats booked.

Each customer's booking has:

Booking ID (integer)Customer Name (string)Number of Tickets (integer)Seat Type (string: "Standard", "Premium", "VIP")

The ticket prices are:

Standard – 250 units per ticketPremium – 400 units per ticketVIP – 600 units per ticket

The calculation rules:

Total Amount = Number of Tickets × Seat Price

If a customer books more than 4 tickets, they get a 10% discount on the total amount.

If the booking is for VIP seats and the total amount exceeds 3000 units, a 5% luxury tax is added after any discount.

Neha has been asked to implement this system using:

A class with attributes for booking details.A constructor to initialize booking details.Getter and Setter methods to retrieve and update booking details if required.A method to calculate the final ticket cost.Objects of the

class to represent bookings.

Finally, display each customer's details and final ticket amount.

Input Format

The first line contains an integer N, representing the number of bookings.

For each booking:

- The next line contains the Booking ID (integer).
- The next line contains the Customer Name (string).
- The next line contains Number of Tickets (integer).
- The next line contains Seat Type ("Standard", "Premium", or "VIP").

Output Format

For each booking, print:

- Booking ID: <booking_id>
- Customer Name: <customer_name>
- Final Ticket Amount: <final_amount> (rounded to one decimal place)

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 1
1001
Ravi Kumar
3
Standard
Output: Booking ID: 1001
Customer Name: Ravi Kumar
Final Ticket Amount: 750.0

Answer

```
import java.util.Scanner;
class Booking {
    private int bookingId;
    private String customerName;
```

```
    private int numberOfTickets;
    private String seatType;
    public Booking(int bookingId, String customerName, int numberOfTickets,
String seatType) {
        this.bookingId = bookingId;
        this.customerName = customerName;
        this.numberOfTickets = numberOfTickets;
        this.seatType = seatType;
    }
    public int getBookingId() {
        return bookingId;
    }
    public String getCustomerName() {
        return customerName;
    }
    public int getNumberOfTickets() {
        return numberOfTickets;
    }
    public String getSeatType() {
        return seatType;
    }
    public void setBookingId(int bookingId) {
        this.bookingId = bookingId;
    }
    public void setCustomerName(String customerName) {
        this.customerName = customerName;
    }
    public void setNumberOfTickets(int numberOfTickets) {
        this.numberOfTickets = numberOfTickets;
    }
    public void setSeatType(String seatType) {
        this.seatType = seatType;
    }
    public double calculateFinalAmount() {
        double seatPrice;
        switch (seatType) {
            case "Standard":
                seatPrice = 250.0;
                break;
            case "Premium":
                seatPrice = 400.0;
                break;
```

```
            case "VIP":
                seatPrice = 600.0;
                break;
            default:
                seatPrice = 0.0;
        }
        double totalAmount = numberOfTickets * seatPrice;
        if (numberOfTickets > 4) {
            totalAmount *= 0.9;
        }
        if (seatType.equals("VIP") && totalAmount > 3000) {
            totalAmount *= 1.05;
        }
        return totalAmount;
    }
}

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        if (!sc.hasNextInt()) {
            System.err.println("No input available for number of bookings.");
            sc.close();
            return;
        }
        int N = sc.nextInt();
        sc.nextLine();
        if (N < 1 || N > 100) {
            System.err.println("Invalid number of bookings: " + N);
            sc.close();
            return;
        }
        for (int i = 0; i < N; i++) {
            if (!sc.hasNextInt()) {
                System.err.println("Insufficient input for booking ID at booking " + (i +
1));
                sc.close();
                return;
            }
            int bookingId = sc.nextInt();
            if (bookingId < 1000 || bookingId > 9999) {
                System.err.println("Invalid booking ID at booking " + (i + 1));
```

```
            sc.close();
            return;
        }
        sc.nextLine();
        if (!sc.hasNextLine()) {
            System.err.println("Insufficient input for customer name at booking " + (i
+ 1));
            sc.close();
            return;
        }
        String customerName = sc.nextLine().trim();
        if (customerName.isEmpty()) {
            System.err.println("Customer name cannot be empty at booking " + (i +
1));
            sc.close();
            return;
        }
        if (!sc.hasNextInt()) {
            System.err.println("Insufficient input for number of tickets at booking " +
(i + 1));
            sc.close();
            return;
        }
        int numberOfTickets = sc.nextInt();
        if (numberOfTickets < 1) {
            System.err.println("Invalid number of tickets at booking " + (i + 1));
            sc.close();
            return;
        }
        sc.nextLine();
        if (!sc.hasNextLine()) {
            System.err.println("Insufficient input for seat type at booking " + (i + 1));
            sc.close();
            return;
        }
        String seatType = sc.nextLine().trim();
        if (!seatType.equals("Standard") && !seatType.equals("Premium") && !
seatType.equals("VIP")) {
            System.err.println("Invalid seat type at booking " + (i + 1));
            sc.close();
            return;
        }
```

```
        Booking booking = new Booking(bookingId, customerName,
numberOfTickets, seatType);
        double finalAmount = booking.calculateFinalAmount();
        System.out.println("Booking ID: " + booking.getBookingId());
        System.out.println("Customer Name: " + booking.getCustomerName());
        System.out.printf("Final Ticket Amount: %.1f\n", finalAmount);
    }
    sc.close();
  }
}
```

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

4. Problem Statement

Each customer at the bank has an Account Number, Customer Name, and an Initial Balance. The bank allows two types of transactions:

Deposit – Increases the balance.Withdrawal – Decreases the balance, but only if enough funds are available. If the withdrawal amount exceeds the available balance, the transaction should be skipped, and the balance should remain unchanged.

You are required to implement this banking system by:

Creating a class with the necessary attributes to store account details.

Using a constructor to initialize the account details when a new account is created.Providing setter methods to update the details if required.Providing getter methods to retrieve account details.Creating objects of this class to represent different customers, where each customer can perform deposits and withdrawals.

Instructions:

Implement the class to store account details.Implement the logic for performing deposit and withdrawal transactions.Ensure that withdrawals don't exceed the available balance.After performing the transactions, print the account number, customer name, and final balance.

Input Format

The first line of input contains an integer N, representing the number of customers.

For each customer:

- The next line contains the account number (integer).
- The following line contains the customer name (string).
- The next line contains the initial balance (double).
- The next line contains the deposit amount (double).
- The next line contains the withdrawal amount (double).

Output Format

For each customer, print the details in the following format:

1. Account Number: <account_number>
2. Customer Name: <customer_name>
3. Final Balance: <final_balance> (rounded to one decimal place)

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 1
1234
Rahul Sharma
5000
2000
3000

Output: Account Number: 1234
Customer Name: Rahul Sharma
Final Balance: 4000.0

Answer

```
import java.util.Scanner;
class Account {
   private int accountNumber;
   private String customerName;
   private double balance;
   public Account(int accountNumber, String customerName, double
initialBalance) {
```

```
        this.accountNumber = accountNumber;
        this.customerName = customerName;
        this.balance = initialBalance;
    }
    public int getAccountNumber() {
        return accountNumber;
    }
    public String getCustomerName() {
        return customerName;
    }
    public double getBalance() {
        return balance;
    }
    public void setAccountNumber(int accountNumber) {
        this.accountNumber = accountNumber;
    }
    public void setCustomerName(String customerName) {
        this.customerName = customerName;
    }
    public void setBalance(double balance) {
        this.balance = balance;
    }
    public void deposit(double amount) {
        if (amount >= 0) {
            balance += amount;
        }
    }
    public void withdraw(double amount) {
        if (amount >= 0 && amount <= balance) {
            balance -= amount;
        }
    }
}
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        if (!sc.hasNextInt()) {
            System.err.println("No input available for number of customers.");
            sc.close();
            return;
        }
        int N = sc.nextInt();
```

```
        sc.nextLine();
        if (N < 1 || N > 100) {
            System.err.println("Invalid number of customers: " + N);
            sc.close();
            return;
        }
        for (int i = 0; i < N; i++) {
            if (!sc.hasNextInt()) {
                System.err.println("Insufficient input for account number at customer "
+ (i + 1));
                sc.close();
                return;
            }
            int accountNumber = sc.nextInt();
            if (accountNumber < 1000 || accountNumber > 9999) {
                System.err.println("Invalid account number at customer " + (i + 1));
                sc.close();
                return;
            }
            sc.nextLine();
            if (!sc.hasNextLine()) {
                System.err.println("Insufficient input for customer name at customer " +
(i + 1));
                sc.close();
                return;
            }
            String customerName = sc.nextLine().trim();
            if (customerName.isEmpty()) {
                System.err.println("Customer name cannot be empty at customer " + (i
+ 1));
                sc.close();
                return;
            }
            if (!sc.hasNextDouble()) {
                System.err.println("Insufficient input for initial balance at customer " + (i
+ 1));
                sc.close();
                return;


}
            double initialBalance = sc.nextDouble();
```

```
        if (initialBalance < 0)
{

            System.err.println("Invalid initial balance at customer " + (i + 1));
            sc.close();
            return;


}
        sc.nextLine(); // Consume newline

        // Read and validate depositAmount
        if (!sc.hasNextDouble())
{

            System.err.println("Insufficient input for deposit amount at customer " +
(i + 1));
            sc.close();
            return;


}
        double depositAmount = sc.nextDouble();
        if (depositAmount < 0)
{

            System.err.println("Invalid deposit amount at customer " + (i + 1));
            sc.close();
            return;


}
        sc.nextLine(); // Consume newline

        // Read and validate withdrawalAmount
        if (!sc.hasNextDouble())
```

```
{
            System.err.println("Insufficient input for withdrawal amount at customer
" + (i + 1));
            sc.close();
            return;

}
        double withdrawalAmount = sc.nextDouble();
        if (withdrawalAmount < 0)
{
            System.err.println("Invalid withdrawal amount at customer " + (i + 1));
            sc.close();
            return;

}
        // Consume newline only if more customers are expected
        if (i < N - 1)
{
            if (sc.hasNextLine())
{
                sc.nextLine();

} else
{
```

```
                System.err.println("Incomplete input for customer " + (i + 1));
                sc.close();
                return;


    }


    }

        // Create Account object
        Account account = new Account(accountNumber, customerName,
initialBalance);

        // Perform transactions
        account.deposit(depositAmount);
        account.withdraw(withdrawalAmount);

        // Print output
        System.out.println("Account Number: " + account.getAccountNumber());
        System.out.println("Customer Name: " + account.getCustomerName());
        System.out.printf("Final Balance: %.1f\n", account.getBalance());


    }

      sc.close();

    }

    }
```

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

5. Problem Statement

Ravi is working as a developer for SecureLogin Systems, which wants to build a system to evaluate the strength of user passwords.

Each user record has:

User ID (integer)User Name (string)Password (string)

The system must calculate whether a password is strong or weak.

A password is considered strong if it meets all of the following conditions:

At least 8 characters long.Contains at least one uppercase letter.Contains at least one lowercase letter.Contains at least one digit.Contains at least one special character (from !@#$%^&*).

Ravi has been asked to implement this system using:

A class with attributes for user details.A constructor to initialize user details.Getter and setter methods to retrieve or update user details.A method to check whether the password is strong.Objects of the class to represent users.

Finally, display each user's details and indicate whether their password is Strong or Weak.

Input Format

The first line contains an integer N, representing the number of users.

For each user:

The next line contains the User ID (integer).

The next line contains the User Name (string).

The next line contains the Password (string).

Output Format

For each user, print the details in the following format:

User ID: <user_id>

User Name: <user_name>

Password: <password>

Password Strength: <Strong/Weak>

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 1
1001
Ravi Kumar
Abc@1234
Output: User ID: 1001
User Name: Ravi Kumar
Password: Abc@1234
Password Strength: Strong

Answer

```
import java.util.Scanner;
class User {
    private int userId;
    private String userName;
    private String password;
    public User(int userId, String userName, String password) {
        this.userId = userId;
        this.userName = userName;
        this.password = password;
    }
    public int getUserId() {
        return userId;
    }
    public String getUserName() {
        return userName;
    }
    public String getPassword() {
        return password;
    }
    public void setUserId(int userId) {
        this.userId = userId;
    }
    public void setUserName(String userName) {
```

```
        this.userName = userName;
    }
    public void setPassword(String password) {
        this.password = password;
    }
    public String checkPasswordStrength() {
        if (password.length() < 8) {
            return "Weak";
        }
        boolean hasUppercase = false;
        boolean hasLowercase = false;
        boolean hasDigit = false;
        boolean hasSpecial = false;
        String specialChars = "!@#$%^&*";
        for (char c : password.toCharArray()) {
            if (Character.isUpperCase(c)) {
                hasUppercase = true;
            } else if (Character.isLowerCase(c)) {
                hasLowercase = true;
            } else if (Character.isDigit(c)) {
                hasDigit = true;
            } else if (specialChars.indexOf(c) != -1) {
                hasSpecial = true;
            }
        }
        if (hasUppercase && hasLowercase && hasDigit && hasSpecial) {
            return "Strong";
        }
        return "Weak";
    }
}
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        if (!sc.hasNextInt()) {
            System.err.println("No input available for number of users.");
            sc.close();
            return;
        }
        int N = sc.nextInt();
        sc.nextLine();
        if (N < 1 || N > 100) {
```

```
            System.err.println("Invalid number of users: " + N);
            sc.close();
            return;
        }
        for (int i = 0; i < N; i++) {
            if (!sc.hasNextInt()) {
                System.err.println("Insufficient input for user ID at user " + (i + 1));
                sc.close();
                return;
            }
            int userId = sc.nextInt();
            if (userId < 1000 || userId > 9999) {
                System.err.println("Invalid user ID at user " + (i + 1));
                sc.close();
                return;
            }
            sc.nextLine();
            if (!sc.hasNextLine()) {
                System.err.println("Insufficient input for user name at user " + (i + 1));
                sc.close();
                return;
            }
            String userName = sc.nextLine().trim();
            if (userName.isEmpty()) {
                System.err.println("User name cannot be empty at user " + (i + 1));
                sc.close();
                return;
            }
            if (!sc.hasNextLine()) {
                System.err.println("Insufficient input for password at user " + (i + 1));
                sc.close();
                return;
            }
            String password = sc.nextLine().trim();
            if (password.isEmpty()) {
                System.err.println("Password cannot be empty at user " + (i + 1));
                sc.close();
                return;
            }
            User user = new User(userId, userName, password);
            System.out.println("User ID: " + user.getUserId());
            System.out.println("User Name: " + user.getUserName());
```

```
            System.out.println("Password: " + user.getPassword());
            System.out.println("Password Strength: " +
user.checkPasswordStrength());
        }
        sc.close();
    }
}
```

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

Rajalakshmi Engineering College

Name: Pratiba S
Email: 241901080@rajalakshmi.edu.in
Roll no: 241901080
Phone: 7200010194
Branch: REC
Department: CSE (CS) - Section 2
Batch: 2028
Degree: B.E - CSE (CS)

Scan to verify results

2024_28_III_OOPS Using Java Lab

REC_2028_OOPS using Java_Week 5_CY

Attempt : 1
Total Mark : 40
Marks Obtained : 40

Section 1 : Coding

1. Problem Statement

You are working as a developer for CityMobile, which wants to build a basic mobile data usage management system.

Each customer has:

A Customer ID (integer)A Customer Name (string)An Initial Data Balance (in GB, double)

The company allows two types of operations:

Recharge – increases the data balance.Usage – decreases the data balance only if enough data is available.

If the usage amount is greater than the available data balance, the usage should not happen, and the balance should remain the same.

You are required to implement this system using:

A class with attributes for customer details.A constructor to initialize customer details.Setter methods to update details if needed.Getter methods to retrieve details.Objects of the class to represent customers.

Finally, display each customer's details after all operations.

Input Format

The first line of input contains an integer N, representing the number of customers.

For each customer:

- The next line contains the Customer ID (integer).
- The following line contains the Customer Name (string).
- The next line contains the Initial Data Balance (double).
- The next line contains the Recharge Amount in GB (double).
- The next line contains the Usage Amount in GB (double).

Output Format

For each customer, print the details in the following format:

Customer ID: <customer_id>

Customer Name: <customer_name>

Final Data Balance: <final_data_balance> GB (The final balance must be rounded to one decimal place.)

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 1
1234
Ravi Kumar
5.0
2.0

3.0
Output: Customer ID: 1234
Customer Name: Ravi Kumar
Final Data Balance: 4.0 GB

Answer

```
import java.util.Scanner;
class Customer {
    private int customerId;
    private String customerName;
    private double dataBalance;
    public Customer(int customerId, String customerName, double dataBalance) {
        this.customerId = customerId;
        this.customerName = customerName;
        this.dataBalance = dataBalance;
    }
    public int getCustomerId() {
        return customerId;
    }
    public String getCustomerName() {
        return customerName;
    }
    public double getDataBalance() {
        return dataBalance;
    }
    public void setCustomerId(int customerId) {
        this.customerId = customerId;
    }
    public void setCustomerName(String customerName) {
        this.customerName = customerName;
    }
    public void setDataBalance(double dataBalance) {
        this.dataBalance = dataBalance;
    }
    public void recharge(double amount) {
        if (amount >= 0) {
            this.dataBalance += amount;
        }
    }
    public void useData(double amount) {
        if (amount <= dataBalance) {
            this.dataBalance -= amount;
```

```
    }
  }
}
public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int N = Integer.parseInt(sc.nextLine());
    for (int i = 0; i < N; i++) {
      int id = Integer.parseInt(sc.nextLine());
      String name = sc.nextLine();
      double initialBalance = Double.parseDouble(sc.nextLine());
      double recharge = Double.parseDouble(sc.nextLine());
      double usage = Double.parseDouble(sc.nextLine());
      Customer customer = new Customer(id, name, initialBalance);
      customer.recharge(recharge);
      customer.useData(usage);
      System.out.println("Customer ID: " + customer.getCustomerId());
      System.out.println("Customer Name: " + customer.getCustomerName());
      System.out.printf("Final Data Balance: %.1f GB%n",
customer.getDataBalance());
    }
    sc.close();
  }
}
```

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

2. Problem Statement

Anjali is now working as a developer for the City Marathon Association, which wants to build a system to track and find the fastest runner among marathon participants.

Each runner’s record has:

Runner ID (integer)Runner Name (string)An array of times (in minutes) taken in 5 marathon events (integers)

The system must calculate:

The average time of each runner (sum of all times / 5).Identify the fastest runner (the one with the lowest average time).If two or more runners have the same average time, the one with the lower Runner ID is considered the fastest runner.

Anjali has been asked to implement this system using:

A class with attributes for runner details.A constructor to initialize runner details.Getter and Setter methods to retrieve and update runner details if required.A method to calculate the average time.Objects of the class to represent runners.

Finally, display each runner's details and announce the Fastest Runner.

Input Format

The first line of input contains an integer N (number of runners).

For each runner:

- The next line contains the Runner ID (integer).
- The following line contains the Runner Name (string).
- The next line contains 5 integers separated by spaces (times in minutes for 5 marathon events).

Output Format

For each runner the output prints the following details:

- Runner ID: <runner_id>
- Runner Name: <runner_name>
- Average Time: <average_time>

Finally, print "Fastest Runner: <runner_name> with <average_time> minutes"

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 1
1001
Ravi Kumar
240 250 245 255 260

Output: Runner ID: 1001
Runner Name: Ravi Kumar
Average Time: 250
Fastest Runner: Ravi Kumar with 250 minutes

Answer

```
import java.util.Scanner;
import java.util.ArrayList;
class Runner {
    private int runnerId;
    private String runnerName;
    private int[] times;
    public Runner(int runnerId, String runnerName, int[] times) {
        this.runnerId = runnerId;
        this.runnerName = runnerName;
        this.times = times;
    }
    public int getRunnerId() {
        return runnerId;
    }
    public String getRunnerName() {
        return runnerName;
    }
    public int[] getTimes() {
        return times;
    }
    public void setRunnerId(int runnerId) {
        this.runnerId = runnerId;
    }
    public void setRunnerName(String runnerName) {
        this.runnerName = runnerName;
    }
    public void setTimes(int[] times) {
        this.times = times;
    }
    public double calculateAverageTime() {
        int total = 0;
        for (int time : times) {
```

```
            total += time;
        }
        return total / 5.0;
    }
}
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        if (!sc.hasNextInt()) {
            System.err.println("No input available for number of runners.");
            sc.close();
            return;
        }
        int N = sc.nextInt();
        sc.nextLine();
        if (N < 1 || N > 50)

{


            System.err.println("Invalid number of runners: " + N);
            sc.close();
            return;


}

        // List to store valid runners
        ArrayList<Runner> runners = new ArrayList<>();

        // Read runner details
        for (int i = 0; i < N; i++)

{


            // Read and validate runnerId
            if (!sc.hasNextInt())

{
```

```
            System.err.println("Insufficient input for runner ID at runner " + (i + 1));
            sc.nextLine(); // Consume any remaining input
            continue;

        }
        int runnerId = sc.nextInt();
        if (runnerId < 1000 || runnerId > 9999)

        {

            System.err.println("Invalid runner ID at runner " + (i + 1));
            sc.nextLine(); // Consume any remaining input
            continue;

        }
        sc.nextLine(); // Consume newline

        // Read and validate runnerName
        if (!sc.hasNextLine())

        {

            System.err.println("Insufficient input for runner name at runner " + (i +
1));
            continue;

        }
        String runnerName = sc.nextLine().trim();
        if (runnerName.isEmpty())

        {

            System.err.println("Runner name cannot be empty at runner " + (i + 1));
            continue;
```

```
}
        // Read and validate 5 times
        int[] times = new int[5];
        boolean validTimes = true;
        for (int j = 0; j < 5; j++)
{
            if (!sc.hasNextInt())
{
                System.err.println("Insufficient input for time " + (j + 1) + " at runner "
+ (i + 1));
                validTimes = false;
                break;
}
            times[j] = sc.nextInt();
            if (times[j] <= 0)
{
                System.err.println("Invalid time " + (j + 1) + " at runner " + (i + 1));
                validTimes = false;
                break;
}
}
        if (!validTimes)
{
```

```
            sc.nextLine(); // Consume any remaining input
            continue;

    }

        // Consume newline only if more runners are expected
        if (i < N - 1)

    {

            if (sc.hasNextLine())

    {

                sc.nextLine();

    } else

    {

                System.err.println("Incomplete input for runner " + (i + 1));
                continue;

    }

    }

        // Add valid runner to list
        runners.add(new Runner(runnerId, runnerName, times));

    }

        // If no valid runners, exit
```

```
        if (runners.isEmpty())
{


            System.err.println("No valid runners to process.");
            sc.close();
            return;


}

        // Print runner details
        for (Runner runner : runners)
{


            System.out.println("Runner ID: " + runner.getRunnerId());
            System.out.println("Runner Name: " + runner.getRunnerName());
            System.out.printf("Average Time: %.0f\n", runner.calculateAverageTime());


}

        // Find fastest runner
        Runner fastestRunner = runners.get(0);
        double minAverageTime = fastestRunner.calculateAverageTime();
        for (Runner runner : runners)
{


            double currentAverageTime = runner.calculateAverageTime();
            if (currentAverageTime < minAverageTime ||
                (currentAverageTime == minAverageTime && runner.getRunnerId() <
fastestRunner.getRunnerId()))

{


                fastestRunner = runner;
```

```
            minAverageTime = currentAverageTime;

        }


    }

        // Print fastest runner
        System.out.printf("Fastest Runner: %s with %.0f minutes\n",
fastestRunner.getRunnerName(), minAverageTime);

        sc.close();

    }

}
```

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

3. Problem Statement

Arjun is working as a developer for CityWater Supply Board, which wants to build a household water billing system.

Each household’s water account has:

A Customer ID (integer)A Customer Name (string)Liters Consumed (double)

The water bill is calculated based on these rules:

For the first 500 liters 2 per literFor the next 500 liters (501–1000) 3 per literFor liters above 1000 5 per literIf the total bill exceeds 3000, a 10% discount is applied on the final bill.

Arjun has been asked to implement this system using:

A class with attributes for customer details.A constructor to initialize customer details.Setter methods to update details if needed.Getter

methods to retrieve details.Objects of the class to represent customers.

Finally, display each customer's details and final bill amount.

Input Format

The first line of input contains an integer N, representing the number of customers.

For each customer:

- The next line contains the Customer ID (integer).
- The following line contains the Customer Name (string).
- The next line contains the Liters Consumed (double).

Output Format

For each customer, print the details in the following format:

Customer ID: <customer_id>

Customer Name: <customer_name>

Final Bill: <final_bill> (rounded to one decimal place)

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 1
1001
Ravi Kumar
300

Output: Customer ID: 1001
Customer Name: Ravi Kumar
Final Bill: 600.0

Answer

```
import java.util.Scanner;

class Customer
```

```
{


    private int customerId;
    private String customerName;
    private double litersConsumed;

    // Constructor
    public Customer(int customerId, String customerName, double
litersConsumed)

{


        this.customerId = customerId;
        this.customerName = customerName;
        this.litersConsumed = litersConsumed;


}

    // Getter methods
    public int getCustomerId()

{


        return customerId;


}

    public String getCustomerName()

{


        return customerName;

}
```

```
    public double getLitersConsumed()
    {

        return litersConsumed;

    }

    // Setter methods
    public void setCustomerId(int customerId)
    {

        this.customerId = customerId;

    }

    public void setCustomerName(String customerName)

    {

        this.customerName = customerName;

    }

    public void setLitersConsumed(double litersConsumed)

    {

        this.litersConsumed = litersConsumed;

    }
```

```
    // Method to calculate final bill
    public double calculateFinalBill()
{

        double totalBill = 0.0;

        // Calculate bill based on tiered pricing
        if (litersConsumed <= 500)

{

            totalBill = litersConsumed * 2.0;

} else if (litersConsumed <= 1000)

{

            totalBill = (500 * 2.0) + ((litersConsumed - 500) * 3.0);

} else

{

            totalBill = (500 * 2.0) + (500 * 3.0) + ((litersConsumed - 1000) * 5.0);

}

        // Apply 10% discount if total bill exceeds   3000
        if (totalBill > 3000)

{

            totalBill *= 0.9;
```

```
}
        return totalBill;


}

}
public class Main
{

   public static void main(String[] args)
{


        Scanner sc = new Scanner(System.in);

        // Read number of customers
        if (!sc.hasNextInt())
{


            System.err.println("No input available for number of customers.");
            sc.close();
            return;


}
        int N = sc.nextInt();
        sc.nextLine(); // Consume newline

        // Validate N
        if (N < 1 || N > 100)
{
```

```
            System.err.println("Invalid number of customers: " + N);
            sc.close();
            return;


    }

        // Process each customer
        for (int i = 0; i < N; i++)

    {


            // Read and validate customerId
            if (!sc.hasNextInt())

    {


                System.err.println("Insufficient input for customer ID at customer " + (i +
    1));
                sc.close();
                return;


    }
            int customerId = sc.nextInt();
            if (customerId < 1000 || customerId > 9999)

    {


                System.err.println("Invalid customer ID at customer " + (i + 1));
                sc.close();
                return;


    }
            sc.nextLine(); // Consume newline
```

```
        // Read and validate customerName
        if (!sc.hasNextLine())
{


            System.err.println("Insufficient input for customer name at customer " +
(i + 1));
            sc.close();
            return;


}
        String customerName = sc.nextLine().trim();
        if (customerName.isEmpty())
{


            System.err.println("Customer name cannot be empty at customer " + (i
+ 1));
            sc.close();
            return;


}

        // Read and validate litersConsumed
        if (!sc.hasNextDouble())
{


            System.err.println("Insufficient input for liters consumed at customer " +
(i + 1));
            sc.close();
            return;


}
        double litersConsumed = sc.nextDouble();
        if (litersConsumed < 0)
```

```
{
            System.err.println("Invalid liters consumed at customer " + (i + 1));
            sc.close();
            return;

}
        // Consume newline only if more customers are expected
        if (i < N - 1)
{
            if (sc.hasNextLine())
{
                sc.nextLine();

} else
{
                System.err.println("Incomplete input for customer " + (i + 1));
                sc.close();
                return;

}

}
        // Create Customer object
        Customer customer = new Customer(customerId, customerName,
litersConsumed);
```

```
        // Print output
        System.out.println("Customer ID: " + customer.getCustomerId());
        System.out.println("Customer Name: " + customer.getCustomerName());
        System.out.printf("Final Bill: %.1f\n", customer.calculateFinalBill());


}

    sc.close();


}

}
```

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

4. Problem Statement

Anjali is working as a developer for the City Basketball Association, which wants to build a system to track and find the top scorer among basketball players.

Each player's record has:

Player ID (integer)Player Name (string)An array of points scored in 5 matches (integers)

The system must calculate:

The total score of each player (sum of all match points).Identify the highest scorer among all players.If two or more players have the same total score, the one with the lower Player ID is considered the top scorer.

Anjali has been asked to implement this system using:

A class with attributes for player details.A constructor to initialize player details.Getter and Setter methods to retrieve and update player details if required.A method to calculate the total score.Objects of the class to

represent players.

Finally, display each player's details and announce the Top Scorer.

Input Format

The first line of input contains an integer N (number of players).

For each player:

- The next line contains the Player ID (integer).
- The following line contains the Player Name (string).
- The next line contains 5 integers separated by spaces (points scored in 5 matches).

Output Format

For each player the output prints the following details:

- Player ID: <player_id>
- Player Name: <player_name>
- Total Score: <total_score>

Finally, print "Top Scorer: <player_name> with <total_score> points"

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 1
1001
Ravi Kumar
10 20 30 40 50

Output: Player ID: 1001
Player Name: Ravi Kumar
Total Score: 150
Top Scorer: Ravi Kumar with 150 points

Answer

```
import java.util.Scanner;
import java.util.ArrayList;

class Player

{


   private int playerId;
   private String playerName;
   private int[] points;

   // Constructor
   public Player(int playerId, String playerName, int[] points)

{


      this.playerId = playerId;
      this.playerName = playerName;
      this.points = points;


}

   // Getter methods
   public int getPlayerId()

{


      return playerId;


}

   public String getPlayerName()

{


      return playerName;
```

```
}
    public int[] getPoints()
{
        return points;
}
    // Setter methods
    public void setPlayerId(int playerId)
{
        this.playerId = playerId;
}
    public void setPlayerName(String playerName)
{
        this.playerName = playerName;
}
    public void setPoints(int[] points)
{
        this.points = points;
```

```
}
    // Method to calculate total score
    public int calculateTotalScore()
{
        int total = 0;
        for (int point : points)
{
            total += point;
}
        return total;
}
}
public class Main
{
    public static void main(String[] args)
{
        Scanner sc = new Scanner(System.in);
        // Read number of players
        if (!sc.hasNextInt())
{
```

```
            System.err.println("No input available for number of players.");
            sc.close();
            return;


    }
        int N = sc.nextInt();
        sc.nextLine(); // Consume newline

        // Validate N
        if (N < 1 || N > 50)
    {


            System.err.println("Invalid number of players: " + N);
            sc.close();
            return;


    }

        // List to store valid players
        ArrayList<Player> players = new ArrayList<>();

        // Read player details
        for (int i = 0; i < N; i++)
    {


            // Read and validate playerId
            if (!sc.hasNextInt())
    {


                System.err.println("Insufficient input for player ID at player " + (i + 1));
                sc.nextLine(); // Consume any remaining input
                continue;
```

```
}
        int playerId = sc.nextInt();
        if (playerId < 1000 || playerId > 9999)

{


            System.err.println("Invalid player ID at player " + (i + 1));
            sc.nextLine(); // Consume any remaining input
            continue;


}
        sc.nextLine(); // Consume newline

        // Read and validate playerName
        if (!sc.hasNextLine())

{


            System.err.println("Insufficient input for player name at player " + (i +
1));
            continue;


}
        String playerName = sc.nextLine().trim();
        if (playerName.isEmpty())

{


            System.err.println("Player name cannot be empty at player " + (i + 1));
            continue;


}

        // Read and validate 5 points
```

```
        int[] points = new int[5];
        boolean validPoints = true;
        for (int j = 0; j < 5; j++)

{


            if (!sc.hasNextInt())

{


                System.err.println("Insufficient input for point " + (j + 1) + " at player "
+ (i + 1));
                validPoints = false;
                break;


}
            points[j] = sc.nextInt();
            if (points[j] < 0)

{


                System.err.println("Invalid point " + (j + 1) + " at player " + (i + 1));
                validPoints = false;
                break;


}


}


        if (!validPoints)

{


            sc.nextLine(); // Consume any remaining input
            continue;
```

```
}
        // Consume newline only if more players are expected
        if (i < N - 1)
{
            if (sc.hasNextLine())
{
                sc.nextLine();
} else
{
                System.err.println("Incomplete input for player " + (i + 1));
                continue;
}
}
        // Add valid player to list
        players.add(new Player(playerId, playerName, points));
}
      // If no valid players, exit
      if (players.isEmpty())
{
```

```
            System.err.println("No valid players to process.");
            sc.close();
            return;


    }

        // Print player details
        for (Player player : players)

    {


            System.out.println("Player ID: " + player.getPlayerId());
            System.out.println("Player Name: " + player.getPlayerName());
            System.out.println("Total Score: " + player.calculateTotalScore());


    }

        // Find top scorer
        Player topScorer = players.get(0);
        int maxScore = topScorer.calculateTotalScore();
        for (Player player : players)

    {


            int currentScore = player.calculateTotalScore();
            if (currentScore > maxScore ||
                (currentScore == maxScore && player.getPlayerId() <
    topScorer.getPlayerId()))

    {


                topScorer = player;
                maxScore = currentScore;
```

```
    }

    }

        // Print top scorer
        System.out.println("Top Scorer: " + topScorer.getPlayerName() + " with " +
    maxScore + " points");

        sc.close();


    }

    }
```

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