Programming Test Results (With Test Cases)

Result Summary

Field	Value
Test ID	41407
Student ID	29015
Programs (with test cases)	3
Total Test Cases	9
Test Cases Passed	9
Fully Passed Programs	3
Partially Passed Programs	0
Failed Programs	0
Overall % (with test cases)	100.00%
Grade	Outstanding

Programs With Test Cases

#	Program Name	Total TC	Passed	Success Rate	Score /10	Submitted At	Attempts
1	Bank Interest Calculation	3	3	100.0%	10	31/1/2026, 3:56:10 pm	0
2	PlayerInfo	2	2	100.0%	10	31/1/2026, 3:55:59 pm	0
3	PayrollManagementSystem	4	4	100.0%	10	31/1/2026, 3:55:33 pm	0

Program Details (With Test Cases)

Program 1: Bank Interest Calculation

Languages:	Java	
Score (010):	10 / 10	
Test Case Summary:	Total: 3	Passed: 3
	Failed: 0	Success: 100.0%
Attempts:	0	
Submitted At:	31/1/2026, 3:56:10 pm	

Description:

A bank wants to build a system to calculate the final payable amount for customers after applying a variable interest rate based on the number of months.

Your task is to design a program using Encapsulation and a Copy Constructor to demonstrate how the final amount changes in a copied object after interest calculation while the original object remains unchanged.

Class Description

Class: BankAccount

Encapsulated Fields (private):

accountHolderName : String

principalAmount : double

timeInMonths : double

interestRate : double

finalAmount : double

Constructors:

Parameterized Constructor
Initializes all the above fields.

Copy Constructor
Takes another BankAccount object as an argument.
Calculates the updated final amount after adding interest to the principal.
Copies all other data (like name, rate, time, etc.) from the original object.

Methods:
public double calculateInterest()

Calculates simple interest based on time and rate:

interest = (principalAmount * interestRate * timeInMonths) / 100

Returns the calculated interest.

public double getFinalAmount()

Returns the principal + interest.

public String toString()

Returns formatted account details.

Interest Rate Logic

If time 1 month rate = 20%

If time > 1 and 1.5 months rate = 22%

If time > 1.5 months rate = 25%

Input Format
accountHolderName principalAmount timeInMonths

Output Format
```
--- Original Account Details ---
<Original Object Details>

--- Updated Account Details (After Interest Calculation using Copy Constructor) ---
<New Object Details>

Interest Added : <calculated interest>
Final Amount : <new finalAmount>
```

Constraints: -

Sample Input:	Amit 20000 1.5
Sample Output:	--- Original Account Details --- Account Holder : Amit Principal Amount : 20000.0 Time (in months) : 1.5 Interest Rate : 22.0% --- Updated Account Details (After Interest Calculation using Copy Constructor) --- Account Holder : Amit Principal Amount : 20000.0 Time (in months) : 1.5 Interest Rate : 22.0% Interest Added : 6600.0 Final Amount : 26600.0
Explanation:	-

Solution Code

```
import java.util.Scanner;
class BankAccount {
    private String accountHolderName;
    private double principalAmount;
    private double timeInMonths;
    private double interestRate;
    private double finalAmount;

public BankAccount(String accountHolderName, double principalAmount, double timeInMonths)
{
        this.accountHolderName = accountHolderName;
        this.principalAmount = principalAmount;
        this.timeInMonths = timeInMonths;


        if (timeInMonths <= 1) {
            this.interestRate = 20.0;
        } else if (timeInMonths <= 1.5) {
            this.interestRate = 22.0;
        } else {
            this.interestRate = 25.0;
        }

        this.finalAmount = principalAmount;
    }

public BankAccount(BankAccount other) {
        this.accountHolderName = other.accountHolderName;
        this.principalAmount = other.principalAmount;
        this.timeInMonths = other.timeInMonths;
        this.interestRate = other.interestRate;


        double interest = calculateInterest();
        this.finalAmount = this.principalAmount + interest;
    }

public double calculateInterest() {
        return (principalAmount * interestRate * timeInMonths) / 100;
    }
public double getFinalAmount() {
        return finalAmount;
    }

public String toString() {
return "Account Holder : " + accountHolderName + "\n" +
"Principal Amount : " + principalAmount + "\n" +
"Time (in months) : " + timeInMonths + "\n" +
"Interest Rate : " + interestRate + "%";
}
public double getInterest() {
return calculateInterest();
}
}

public class Bank_Interest_Calculation {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        String accountHolderName = sc.next();
        double principalAmount = sc.nextDouble();
        double timeInMonths = sc.nextDouble();

        BankAccount original = new BankAccount(accountHolderName, principalAmount,
timeInMonths);

        IO.println("--- Original Account Details ---");
        IO.println(original.toString());
        BankAccount copy = new BankAccount(original);



        IO.println("\n--- Updated Account Details (After Interest Calculation using Copy
Constructor) ---");
        IO.println(copy.toString());
        IO.println("Interest Added : " + copy.getInterest());
        IO.println("Final Amount : " + copy.getFinalAmount());
    }
}
```

Program 2: PlayerInfo

Languages:	Java	
Score (010):	10 / 10	
Test Case Summary:	Total: 2	Passed: 2
	Failed: 0	Success: 100.0%
Attempts:	0	
Submitted At:	31/1/2026, 3:55:59 pm	

Description:

Create a Java program to demonstrate Encapsulation using a Player class.
The program should calculate the players Batting Average and Strike Rate, and categorize their performance based on the strike rate.

All player details must be private and accessible only through public setter and getter methods, demonstrating data hiding.

Scenario

Every cricket players batting data is recorded after several innings.
You have to build a class that takes the following inputs:

Player Name

Total Runs Scored

Total Balls Faced

Total Innings Played

Times Player Got Out

You must calculate:

Batting Average = Total Runs / Times Out

Strike Rate = (Total Runs / Total Balls) 100

Then categorize the players performance:

Strike Rate   Category

150	Explosive
100	Good
< 100	Needs Improvement

If Times Out = 0, display "Infinity" as the batting average (dont use exceptions).

Formula Reference

Batting Average = totalRuns / timesOut

Strike Rate = (totalRuns / totalBalls) * 100

Encapsulation Requirements

All data members must be private.

Use public setter methods to assign values.

Use public getter methods to fetch computed results.

Display all details using the toString() method.

Input Format

```
-----------------
<playerName> <totalRuns> <totalBalls> <innings> <timesOut>
```

Output Format

```
----------------------
Player Performance Summary:
Player Name : <name>
Total Runs  : <runs>
Balls Faced : <balls>
Innings     : <innings>
Times Out   : <timesOut>
Batting Avg : <average>
Strike Rate : <strikeRate>
Performance : <category>
```

Constraints: 0 totalRuns 10000 0 totalBalls 10000 1 innings 500 0 timesOut innings If totalBalls = 0, strike rate = 0 If timesOut = 0, batting average = "Infinity"

Sample Input: Virat 250 150 5 5

Sample Output: Player Performance Summary: Player Name : Virat Total Runs : 250 Balls Faced : 150 Innings : 5 Times Out : 5 Batting Avg : 50.0 Strike Rate : 166.67 Performance : Explosive

Explanation: -

Solution Code

```
import java.util.Scanner;
class Player
{
private String playerName;
private int totalRuns;
private int totalBalls;
private int innings;
private int timesOut;

public void setPlayerName(String playerName){
this.playerName = playerName;
}
public void setTotalRuns(int totalRuns){
this.totalRuns = totalRuns;
}
public void setTotalBalls(int totalBalls){
this.totalBalls = totalBalls;
}
public void setInnings(int innings){
this.innings = innings;
}
public void setTimesOut(int timesOut){
this.timesOut = timesOut;
}



public String getPlayerName(){
return playerName;
}
public int getTotalRuns(){
return totalRuns;
}
public int getTotalBalls(){
return totalBalls;
}
public int getInnings(){
return innings;
}
public int getTimesOut(){
```

```
return timesOut;
}



public String getBattingAvg(){
if (timesOut == 0) {
return "Infinity";
}
return String.format("%.2f", (double) totalRuns / timesOut);
}
public double getStrikeRate() {
if (totalBalls == 0) {
return 0.0;
}
return ((double) totalRuns / totalBalls) * 100;
}

public String getPerformance() {
double strikeRate = getStrikeRate();
if (strikeRate >= 150) {
return "Explosive";
} else if (strikeRate >= 100) {
return "Good";
} else {
return "Needs Improvement";
}
}

public String toString() {
return "Player Performance Summary:\n" +
"Player Name : " + playerName + "\n" +
"Total Runs  : " + totalRuns + "\n" +
"Balls Faced : " + totalBalls + "\n" +
"Innings     : " + innings + "\n" +
"Times Out   : " + timesOut + "\n" +
"Batting Avg : " + getBattingAvg() + "\n" +
"Strike Rate : " + String.format("%.2f", getStrikeRate()) + "\n" +
"Performance : " + getPerformance();
  }
}

public class PlayerInfo {
```

```
public static void main(String[] args) {
Scanner sc = new Scanner(System.in);

Player player = new Player();
player.setPlayerName(sc.next());
player.setTotalRuns(sc.nextInt());
player.setTotalBalls(sc.nextInt());
player.setInnings(sc.nextInt());
player.setTimesOut(sc.nextInt());
System.out.println(player.toString());
sc.close();
 }
}
```

Program 3: PayrollManagementSystem

Languages:	Java	
Score (010):	10 / 10	
Test Case Summary:	Total: 4	Passed: 4
	Failed: 0	Success: 100.0%
Attempts:	0	
Submitted At:	31/1/2026, 3:55:33 pm	

Description:

The payroll system of an organization involves calculating the gross salary of each type of employee and the tax applicable to each.

Note : Take parameterized constructor to initialize all fields.
Attributes must be private for all the BLC classes.

Create the following BLC classes as described below.

Class Employee
Fields: employeeId: int, employeeName : String, basicSalary : double, HRAPer : double, DAPer : double

Public Method: calculateGrossSalary() - returns a double
Calculate the gross salary as : basicSalary +HRAPer +DAPer

Class Manager
Fields: managerId: int, managerName : String, basicSalary : double, HRAPer : double,DAPer : double, projectAllowance: double

public Method: calculateGrossSalary() - returns a double

Calculate the gross salary as : basicSalary +HRAPer +DAPer + projectAllowance

Class Trainer
Fields: trainerId: int, trainerName : String, basicSalary : double, HRAPer : double,DAPer : double, batchCount: int, perkPerBatch: double

public Method: calculateGrossSalary() - returns a double
Calculate the gross salary as : basicSalary +HRAPer +DAPer +(batchCount * perkPerBatch)

Class Sourcing
Fields: sourceId: int, sourcaName : String, basicSalary : double, HRAPer : double,DAPer : double, enrollmentTarget: int, enrollmentReached: int, perkPerEnrollment: double
Public Method: calculateGrossSalary() - returns a double

Calculate the gross salary as : basicSalary +HRAPer +DAPer +(((enrollmentReached/enrollmentTarget)*100)*perkPerEnrollment)

Class TaxUtil
Fields: None
Public Methods:
calculateTax(Employee e) - returns a double
calculateTax(Manager m) - returns a double
calculateTax(Trainer t) - returns a double
calculateTax(Sourcing s) - returns a double

Tax Calculation Logic: If gross salary is greater than 50000 tax is 20% else, tax is 5%.

An ELC class TaxCalculation is given to you with the main Method. Use this class to test your solution.

Constraints: -

Sample Input: 1 101 Ravi 30000 5000 4000

Sample Output: Gross Salary : 39000.0 Tax : 1950.0

Explanation: Assumption for Input First input decides the employee type: 1 Employee 2 Manager 3 Trainer 4 Sourcing TEST CASE 1 Employee Input 1 101 Ravi 30000 5000 4000 Calculation Gross Salary = 30000 + 5000 + 4000 = 39000 Tax = 5% of 39000 = 1950 Output Gross Salary : 39000.0 Tax : 1950.0 TEST CASE 2 Manager Input 2 201 Anitha 45000 7000 6000 5000 Calculation Gross Salary = 45000 + 7000 + 6000 + 5000 = 63000 Tax = 20% of 63000 = 12600 Output Gross Salary : 63000.0 Tax : 12600.0 TEST CASE 3 Trainer Input 3 301 Suresh 35000 6000 5000 4 2000 Calculation Batch Perk = 4 2000 = 8000 Gross Salary = 35000 + 6000 + 5000 + 8000 = 54000 Tax = 20% of 54000 = 10800 Output Gross Salary : 54000.0 Tax : 10800.0 TEST CASE 4 Sourcing Input 4 401 Kiran 28000 4000 3000 100 80 100 Calculation Achievement % = (80 / 100) 100 = 80 Perk = 80 100 = 8000 Gross Salary = 28000 + 4000 + 3000 + 8000 = 43000 Tax = 5% of 43000 = 2150 Output Gross Salary : 43000.0 Tax : 2150.0

Solution Code

```
import java.util.Scanner;

//Employee class
class Employee {
    private int employeeId;
    private String employeeName;
    private double basicSalary;
    private double HRAPer;
    private double DAPer;

    public Employee(int employeeId, String employeeName, double basicSalary, double
HRAPer, double DAPer) {
        this.employeeId = employeeId;
        this.employeeName = employeeName;
        this.basicSalary = basicSalary;
        this.HRAPer = HRAPer;
        this.DAPer = DAPer;
    }

    public double calculateGrossSalary() {
        return basicSalary + HRAPer + DAPer;
    }
}

// Manager class
class Manager {
    private int managerId;
    private String managerName;
    private double basicSalary;
    private double HRAPer;
    private double DAPer;
    private double projectAllowance;
    public Manager(int managerId, String managerName, double basicSalary, double HRAPer,
double DAPer, double projectAllowance) {
        this.managerId = managerId;
        this.managerName = managerName;
        this.basicSalary = basicSalary;
        this.HRAPer = HRAPer;
        this.DAPer = DAPer;
        this.projectAllowance = projectAllowance;
    }

    public double calculateGrossSalary() {
        return basicSalary + HRAPer + DAPer + projectAllowance;
    }
}

// Trainer class
class Trainer {
    private int trainerId;
    private String trainerName;
    private double basicSalary;
    private double HRAPer;
    private double DAPer;
    private int batchCount;
    private double perkPerBatch;

    public Trainer(int trainerId, String trainerName, double basicSalary, double HRAPer,
double DAPer, int batchCount, double perkPerBatch) {
        this.trainerId = trainerId;
        this.trainerName = trainerName;
        this.basicSalary = basicSalary;
        this.HRAPer = HRAPer;
        this.DAPer = DAPer;
        this.batchCount = batchCount;
        this.perkPerBatch = perkPerBatch;
    }

    public double calculateGrossSalary() {
        return basicSalary + HRAPer + DAPer + (batchCount * perkPerBatch);
    }
}

// Sourcing class
```

```
class Sourcing {
    private int sourceId;
    private String sourcaName;
    private double basicSalary;
    private double HRAPer;
    private double DAPer;
    private int enrollmentTarget;
    private int enrollmentReached;
    private double perkPerEnrollment;

    public Sourcing(int sourceId, String sourcaName, double basicSalary, double HRAPer,
double DAPer, int enrollmentTarget, int enrollmentReached, double perkPerEnrollment) {
        this.sourceId = sourceId;
        this.sourcaName = sourcaName;
        this.basicSalary = basicSalary;
        this.HRAPer = HRAPer;
        this.DAPer = DAPer;
        this.enrollmentTarget = enrollmentTarget;
        this.enrollmentReached = enrollmentReached;
        this.perkPerEnrollment = perkPerEnrollment;
    }

    public double calculateGrossSalary() {
        return basicSalary + HRAPer + DAPer + (((double)enrollmentReached /
enrollmentTarget) * 100 * perkPerEnrollment);
    }
}

// TaxUtil class
class TaxUtil {
    public double calculateTax(Employee e) {
        double grossSalary = e.calculateGrossSalary();
        if (grossSalary > 50000) {
            return grossSalary * 0.20;
        } else {
            return grossSalary * 0.05;
        }
    }

    public double calculateTax(Manager m) {
        double grossSalary = m.calculateGrossSalary();
        if (grossSalary > 50000) {
            return grossSalary * 0.20;
        } else {
            return grossSalary * 0.05;
        }
    }

    public double calculateTax(Trainer t) {
        double grossSalary = t.calculateGrossSalary();
        if (grossSalary > 50000) {
            return grossSalary * 0.20;
        } else {
            return grossSalary * 0.05;
        }
    }

    public double calculateTax(Sourcing s) {
        double grossSalary = s.calculateGrossSalary();
        if (grossSalary > 50000) {
            return grossSalary * 0.20;
        } else {
            return grossSalary * 0.05;
        }
    }
}

// Main class
public class TaxCalculation {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int type = sc.nextInt();
        TaxUtil taxUtil = new TaxUtil();
        double grossSalary = 0;
        double tax = 0;

        if (type == 1) {
            int employeeId = sc.nextInt();
            String employeeName = sc.next();
            double basicSalary = sc.nextDouble();
            double HRAPer = sc.nextDouble();
            double DAPer = sc.nextDouble();
            Employee emp = new Employee(employeeId, employeeName, basicSalary, HRAPer,
DAPer);
```

```java
            grossSalary = emp.calculateGrossSalary();
            tax = taxUtil.calculateTax(emp);
        } else if (type == 2) {
            int managerId = sc.nextInt();
            String managerName = sc.next();
            double basicSalary = sc.nextDouble();
            double HRAPer = sc.nextDouble();
            double DAPer = sc.nextDouble();
            double projectAllowance = sc.nextDouble();
            Manager mgr = new Manager(managerId, managerName, basicSalary, HRAPer, DAPer,
projectAllowance);
            grossSalary = mgr.calculateGrossSalary();
            tax = taxUtil.calculateTax(mgr);
        } else if (type == 3) {
            int trainerId = sc.nextInt();
            String trainerName = sc.next();
            double basicSalary = sc.nextDouble();
            double HRAPer = sc.nextDouble();
            double DAPer = sc.nextDouble();
            int batchCount = sc.nextInt();
            double perkPerBatch = sc.nextDouble();
            Trainer trainer = new Trainer(trainerId, trainerName, basicSalary, HRAPer,
DAPer, batchCount, perkPerBatch);
            grossSalary = trainer.calculateGrossSalary();
            tax = taxUtil.calculateTax(trainer);
        } else if (type == 4) {
            int sourceId = sc.nextInt();
            String sourcaName = sc.next();
            double basicSalary = sc.nextDouble();
            double HRAPer = sc.nextDouble();
            double DAPer = sc.nextDouble();
            int enrollmentTarget = sc.nextInt();
            int enrollmentReached = sc.nextInt();
            double perkPerEnrollment = sc.nextDouble();
            Sourcing sourcing = new Sourcing(sourceId, sourcaName, basicSalary, HRAPer,
DAPer, enrollmentTarget, enrollmentReached, perkPerEnrollment);
            grossSalary = sourcing.calculateGrossSalary();
            tax = taxUtil.calculateTax(sourcing);
        }

        System.out.println("Gross Salary : " + grossSalary);
        System.out.println("Tax : " + tax);
        sc.close();
    }
}
```