**Constructor execution hierarchy, Superclass References and Subclass Objects, Method Overriding, Abstract Classes, Using final**.

****

****

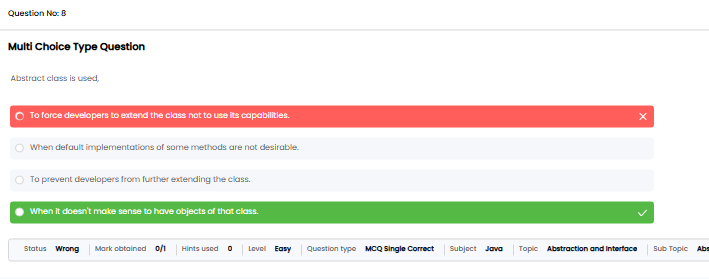
****

****

****

****

****

****

****

****

**Single File Programming Question**

**Problem Statement**

Jessica is tasked with designing a fantasy game character system. The system includes an abstract class named **GameCharacter** with two abstract methods: **attack()** and **defend()**.

Two subclasses, **Warrior** and **Wizard,** extend the GameCharacter class.

The program prompts the player to choose a character class (1. Warrior, 2. Wizard) and input their character's strength or magic power. The dynamic calculations involve tripling the strength (strength \* 3) for the Warrior's attack and doubling the magic power(power \* 2) for the Wizard's attack.

Jessica needs your help in completing the program. Help her finish it.

**Input format :**

The first line of input consists of an integer, representing the choice of the character - 1 for Warrior and 2 for Wizard.

If the choice is 1, the second line consists of an integer **N,** representing the strength.

If the choice is 2, the second line consists of an integer **M,** representing the magic power.

**Output format :**

If the choice is 1, the output displays the actions of a warrior in the following format:

"Warrior Actions:

Attack: Powerful sword slash for [result] damage!

Defend: Raises shield, defence boosted by [N]!"

If the choice is 2, the output displays the actions of a wizard in the following format:

"Wizard Actions:

Attack: Casts spell, deals [result] magical damage!

Defend: Creates magical barrier, defence boosted by [M]!"

If any other choice is given, print "Invalid choice".

**Refer to the sample output for formatting specifications.**

**Code constraints :**

1 ≤ M, N ≤ 106

**Sample test cases :**

**Input 1 :**

1

68

**Output 1 :**

Warrior Actions:

Attack: Powerful sword slash for 204 damage!

Defend: Raises shield, defence boosted by 68!

**Input 2 :**

2

998

**Output 2 :**

Wizard Actions:

Attack: Casts spell, deals 1996 magical damage!

Defend: Creates magical barrier, defence boosted by 998!

**Input 3 :**

3

76

**Output 3 :**

Invalid choice

import java.util.Scanner;

abstract class GameCharacter {

abstract void attack(int strength);

abstract void defend(int defense);

}

class Warrior extends GameCharacter {

void attack(int strength) {

int damage = strength \* 3;

System.out.println("Attack: Powerful sword slash for " + damage + " damage!");

}

void defend(int defense) {

System.out.println("Defend: Raises shield, defence boosted by " + defense + "!");

}

}

class Wizard extends GameCharacter {

void attack(int magicPower) {

int spellPower = magicPower \* 2;

System.out.println("Attack: Casts spell, deals " + spellPower + " magical damage!");

}

void defend(int defense) {

System.out.println("Defend: Creates magical barrier, defence boosted by " + defense + "!");

}

}

class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int choice = scanner.nextInt();

int attributeValue = scanner.nextInt();

GameCharacter character;

if (choice == 1) {

character = new Warrior();

} else if (choice == 2) {

character = new Wizard();

} else {

System.out.println("Invalid choice");

return;

}

System.out.println((choice == 1 ? "Warrior" : "Wizard") + " Actions:");

character.attack(attributeValue);

character.defend(attributeValue);

scanner.close();

}

}

**Problem Statement**

Aravind wants to design a program for an electricity monitor that takes user input for the number of hours to monitor and hourly electricity usage.

Your task is to assist Aravind in creating a **final** class, **ElectricityMonitor**, with a method to analyze patterns and identify the peak usage time. Display the results, indicating the hour and value of peak usage.

**Input format :**

The first line of input consists of an integer **N,**representing the number of hours to monitor.

The second line consists of **N** space-separated double values, representing the electricity usage for each hour.

**Output format :**

The output prints the hour and the peak usage time, rounded off to two decimal places.

**Refer to the sample output for the exact text.**

**Code constraints :**

The given test cases fall under the following constraints:

2 ≤ N ≤ 10

1.0 ≤ electricity usage ≤ 500.0

**Sample test cases :**

**Input 1 :**

5

150.0 125.5 141.6 129.8 114.8

**Output 1 :**

Peak Usage Time: Hour 1 with 150.00 kWh

**Input 2 :**

8

4.92 8.64 8.73 9.46 5.78 4.92 5.64 9.45

**Output 2 :**

Peak Usage Time: Hour 4 with 9.46 kWh

**Input 3 :**

3

137.5 127.8 145.2

**Output 3 :**

Peak Usage Time: Hour 3 with 145.20 kWh

import java.util.Scanner;

final class ElectricityMonitor

{

private double[] hourlyUsage;

public ElectricityMonitor(double[] hourlyUsage) {

this.hourlyUsage = hourlyUsage;

}

public void analyzeUsagePatterns() {

double maxUsage = 0;

int peakHour = -1;

for (int hour = 0; hour < hourlyUsage.length; hour++) {

if (hourlyUsage[hour] > maxUsage) {

maxUsage = hourlyUsage[hour];

peakHour = hour;

}

}

System.out.print("Peak Usage Time: ");

if (peakHour != -1) {

System.out.printf("Hour %d with %.2f kWh", peakHour + 1, maxUsage);

}

}

}public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int numHours = scanner.nextInt();

double[] hourlyUsage = new double[numHours];

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

hourlyUsage[i] = scanner.nextDouble();

}

ElectricityMonitor electricityMonitor = new ElectricityMonitor(hourlyUsage);

electricityMonitor.analyzeUsagePatterns();

scanner.close();

}

}

**Problem Statement**

Teena's retail store has implemented a Loyalty Points System to reward customers based on their spending. The program includes two classes: **Customer** and **PremiumCustomer**.

For regular customers: Loyalty points = amount spent / 10

For premium customers: Loyalty points = 2 \* (amount spent / 10)

Calculate and display the loyal points received by the customers using an overridden method **calculateLoyaltyPoints.**

**Input format :**

The first line of input consists of an integer representing the amount spent by the customer.

The second line consists of premium customer status (a string) - "yes" if the customer is a premium customer, "no" if they are not.

**Output format :**

The output displays the loyalty points earned based on the amount spent.

**Refer to the sample output for formatting specifications.**

**Code constraints :**

1 ≤ amount ≤ 10,000

**Sample test cases :**

**Input 1 :**

50

yes

**Output 1 :**

10

**Input 2 :**

40

no

**Output 2 :**

4

import java.util.Scanner;

class Customer {

public int calculateLoyaltyPoints(int amountSpent) {

return amountSpent / 10;

}

}

class PremiumCustomer extends Customer {

public int calculateLoyaltyPoints(int amountSpent) {

return 2 \* (amountSpent / 10);

}

}

public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int amountSpent = scanner.nextInt();

String isPremium = scanner.next().toLowerCase();

Customer customer;

if (isPremium.equals("yes")) {

customer = new PremiumCustomer();

} else {

customer = new Customer();

}

int loyaltyPoints = customer.calculateLoyaltyPoints(amountSpent);

System.out.println(loyaltyPoints);

}

}

Roshni is tasked with developing a program for concatenating two arrays provided by the user. To accomplish this, she wants to create a class named **ArrayConcatenator**. The class includes a constructor to concatenate the elements of the input arrays. She wants to print the elements of the resulting array.

Help Roshni to complete the program.

**Input format :**

The first line of input consists of an integer **N,**representing the number of elements of the first array.

The second line consists of **N**space-separated integers representing the first array elements.

The third line consists of an integer**M,** representing the number of elements of the second array.

The fourth line consists of **M**space-separated integers, representing the second array elements.

**Output format :**

The output prints the concatenated array of elements separated by space.

**Refer to the sample output for the formatting specifications.**

**Code constraints :**

In this scenario, the test cases fall under the following constraints

1 ≤ N, M ≤ 10

1 ≤ array elements ≤ 100

**Sample test cases :**

**Input 1 :**

5

7 8 5 6 9

3

1 4 3

**Output 1 :**

7 8 5 6 9 1 4 3

**Input 2 :**

4

78 56 93 14

1

56

**Output 2 :**

78 56 93 14 56

import java.util.Scanner;

class ArrayConcatenator {

private int[] concatenatedArray;

public ArrayConcatenator(int[] firstArray, int[] secondArray) {

concatenatedArray = new int[firstArray.length + secondArray.length];

for (int i = 0; i < firstArray.length; i++) {

concatenatedArray[i] = firstArray[i];

}

for (int i = 0; i < secondArray.length; i++) {

concatenatedArray[firstArray.length + i] = secondArray[i];

}

}

public void printConcatenatedArray() {

for (int i = 0; i < concatenatedArray.length; i++) {

System.out.print(concatenatedArray[i] + " ");

}

}

}

class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int size1 = scanner.nextInt();

int[] firstArray = new int[size1];

for (int i = 0; i < size1; i++) {

firstArray[i] = scanner.nextInt();

}

int size2 = scanner.nextInt();

int[] secondArray = new int[size2];

for (int i = 0; i < size2; i++) {

secondArray[i] = scanner.nextInt();

}

ArrayConcatenator arrayConcatenator = new ArrayConcatenator(firstArray, secondArray);

arrayConcatenator.printConcatenatedArray();

}

}**Problem Statement**

Wick is developing a real-time strategy game where the players command armies represented by square matrices. The game requires matrix operations to calculatearmy strength and overall battle outcomes.

Write a program to assist Wich that includes an abstract class **MatrixOperation** with an abstract method **performOperation()** and a class **MatrixAddition.** Calculate the army strength by adding all the elements in the given matrices. Display the matrix that represents the army's strength.

**Input format :**

The first line of input consists of an integer **N,**representing the number of rows and columns of a square matrix.

The next **N** lines consist of **N** space-separated integers, representing the elements of the first matrix.

The following **N** lines consist of **N** space-separated integers representing the elements of another matrix.

**Output format :**

The output prints the army strength, which is the addition of the given matrices.

**Refer to the sample output for formatting specifications.**

**Code constraints :**

The given test cases fall under the following constraints:

1 ≤ N ≤ 5

**Sample test cases :**

**Input 1 :**

2

1 2

3 4

5 6

7 8

**Output 1 :**

6 8

10 12

**Input 2 :**

3

1 2 3

4 5 6

7 8 9

10 11 12

13 14 15

16 17 18

**Output 2 :**

11 13 15

17 19 21

23 25 27

Import java.util.\*;

abstract class MatrixOperation {

abstract int[][] performOperation(int[][] matrix1, int[][] matrix2);

}

class MatrixAddition extends MatrixOperation {

int[][] performOperation(int[][] matrix1, int[][] matrix2) {

int n = matrix1.length;

int[][] result = new int[n][n];

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

for (int j = 0; j < n; j++) {

result[i][j] = matrix1[i][j] + matrix2[i][j];

}

}

return result;

}

}class Main {

private static void inputMatrix(int[][] matrix, Scanner scanner) {

for (int i = 0; i < matrix.length; i++) {

for (int j = 0; j < matrix[0].length; j++) {

matrix[i][j] = scanner.nextInt();

}

}

}

private static void displayMatrix(int[][] matrix) {

for (int[] row : matrix) {

for (int value : row) {

System.out.print(value + " ");

}

System.out.println();

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int n = scanner.nextInt();

int[][] matrix1 = new int[n][n];

int[][] matrix2 = new int[n][n];

inputMatrix(matrix1, scanner);

inputMatrix(matrix2, scanner);

MatrixOperation addition = new MatrixAddition();

int[][] resultAddition = addition.performOperation(matrix1, matrix2);

displayMatrix(resultAddition);

scanner.close();

}

}

**Problem Statement**

Shreya is a student who needs to calculate her exam percentage. Her regular percentage is calculated using the formula (obtained marks / total marks) \* 100. However, as a scholarship student, she receives an additional 5% bonus on her calculated percentage.

Write a program that uses an overridden method **calculatePercentage** to compute both her regular and scholarship percentages.

**Input format :**

The first line of input consists of an integer **n,** which represents the total marks in the exam.

The second line consists of an integer **m,** which represents the marks obtained by the student.

**Output format :**

The first line of output displays a double value, representing the percentage of marks obtained by the regular student, rounded to two decimal places.

The second line displays a double value, representing the percentage of marks obtained by the scholarship student, rounded to two decimal places.

**Refer to the sample output for formatting specifications.**

**Code constraints :**

0 ≤ n ≤ 1200

m < n

**Sample test cases :**

**Input 1 :**

500

450

**Output 1 :**

Student Percentage: 90.00%

Scholarship Student Percentage: 95.00%

**Input 2 :**

600

300

**Output 2 :**

Student Percentage: 50.00%

Scholarship Student Percentage: 55.00%

import java.util.Scanner;

class Student {

public double calculatePercentage(int totalMarks, int obtainedMarks) {

double percentage = ((double) obtainedMarks / totalMarks) \* 100.0;

return percentage;

}

}

class ScholarshipStudent extends Student {

public double calculatePercentage(int totalMarks, int obtainedMarks) {

double basePercentage = super.calculatePercentage(totalMarks, obtainedMarks);

double scholarshipPercentage = basePercentage + 5.0;

return scholarshipPercentage;

}

}

public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

Student student = new Student();

ScholarshipStudent scholarshipStudent = new ScholarshipStudent();

int totalMarks = scanner.nextInt();

int obtainedMarks = scanner.nextInt();

double studentPercentage = student.calculatePercentage(totalMarks, obtainedMarks);

double scholarshipStudentPercentage = scholarshipStudent.calculatePercentage(totalMarks, obtainedMarks);

System.out.printf("Student Percentage: %.2f%%\n", studentPercentage);

System.out.printf("Scholarship Student Percentage: %.2f%%\n", scholarshipStudentPercentage);

scanner.close();

}

}

Imagine Maria is developing a game, where it involves collecting resources on each planet, and the availability of resources is modelled using ageometric progression.

Create an abstract class **Series** with an abstract method **nextTerm().** Implement a subclass called **GeometricSeries** that calculates the next term in a geometric progression for the resource collection on planets. Allow players to input the initial resource level, resource growth ratio, and the number of planets they plan to explore.

Display the predicted resource levels on each planet.

**Input format :**

The first line of input consists of a single positive integer**,**representing the initial resource level.

The secondline consists of a single positive integer, representing the growth ratio.

The third line consists of a single positive integer, representing the number of planets.

**Output format :**

The output displays the resource levels of each planet, separated by space.

**Refer to the sample output for formatting specifications.**

**Code constraints :**

In this scenario, the test cases fall under the following constraints:

1 ≤ resource ≤ 10

1 ≤ growth rate ≤ 10

1 ≤ number of planets ≤ 8

**Sample test cases :**

**Input 1 :**

2

3

5

**Output 1 :**

2 6 18 54 162

**Input 2 :**

5

3

6

**Output 2 :**

5 15 45 135 405 1215

**Input 3 :**

2

9

8

**Output 3 :**

2 18 162 1458 13122 118098 1062882 9565938

import java.util.Scanner;

abstract class Series {

abstract int nextTerm();

}

class GeometricSeries extends Series {

private int firstTerm;

private int commonRatio;

private int numberOfTerms;

private int currentTerm;

public GeometricSeries(int firstTerm, int commonRatio, int numberOfTerms) {

this.firstTerm = firstTerm;

this.commonRatio = commonRatio;

this.numberOfTerms = numberOfTerms;

this.currentTerm = 0;

}

int nextTerm() {

int term = (int) (firstTerm \* Math.pow(commonRatio, currentTerm));

currentTerm++;

return term;

}

}

public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int firstTerm = scanner.nextInt();

int commonRatio = scanner.nextInt();

int numberOfTerms = scanner.nextInt();

GeometricSeries geometricSeries = new GeometricSeries(firstTerm, commonRatio, numberOfTerms);

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

System.out.print(geometricSeries.nextTerm() + " ");

}

scanner.close();

}

}

**Problem Statement**

Mary is managing a business and wants to analyze its profitability. She has a regular business model and a seasonal one. Mary is using a program that calculates and compares the profit margins of both models based on revenue and costs.

The program defines a base class **BusinessUtility** with a method **calculateMargin** for calculating the profit margin and a derived class **SeasonalBusinessUtility** that overrides the margin calculation method to include an additional seasonal adjustment.

.

**Note:** margin = (revenue − cost)/revenue \* 100, seasonal margin = (margin + 10)

**Input format :**

The first line of input consists of a double value **r,** representing the revenue.

The second line consists of a double value **c,** representing the cost.

**Output format :**

The output should display the regular margin and seasonal margin of the company's business.

If the regular margin is less than 10, print "Business is not profitable.". If it is 10 or greater, print "Business is profitable."

Round off both margins to two decimal places.

**Refer to the sample output for formatting specifications.**

**Code constraints :**

In this scenario, the test cases fall under the following constraints:

1.0 ≤ r ≤ 1000.0

1.0 ≤ c ≤ 1000.0

**Sample test cases :**

**Input 1 :**

1000.0

800.0

**Output 1 :**

Regular Margin: 20.00%

Seasonal Margin: 30.00%

Business is profitable.

**Input 2 :**

1000.0

950.0

**Output 2 :**

Regular Margin: 5.00%

Seasonal Margin: 15.00%

Business is not profitable.

import java.util.Scanner;

import java.text.DecimalFormat;

class BusinessUtility {

public double calculateMargin(double revenue, double cost) {

double margin = ((revenue - cost) / revenue) \* 100;

return margin;

}

}

class SeasonalBusinessUtility extends BusinessUtility {

public double calculateMargin(double revenue, double cost) {

double baseMargin = super.calculateMargin(revenue, cost);

double seasonalMargin = baseMargin + 10;

return seasonalMargin;

}

}

class ProfitabilityChecker {

public void checkProfitability(double regularMargin) {

if (regularMargin < 10) {

System.out.println("Business is not profitable.");

} else {

System.out.println("Business is profitable.");

}

}

}

class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

double revenue = scanner.nextDouble();

double cost = scanner.nextDouble();

BusinessUtility business = new BusinessUtility();

SeasonalBusinessUtility seasonalBusiness = new SeasonalBusinessUtility();

double regularMargin = business.calculateMargin(revenue, cost);

double seasonalMargin = seasonalBusiness.calculateMargin(revenue, cost);

DecimalFormat df = new DecimalFormat("#.00");

System.out.println("Regular Margin: " + df.format(regularMargin) + "%");

System.out.println("Seasonal Margin: " + df.format(seasonalMargin) + "%");

ProfitabilityChecker checker = new ProfitabilityChecker();

checker.checkProfitability(regularMargin);

scanner.close();

}

}

Arsh wants to developing a program for an Employee Management System with two classes: **Employee** and **Manager.**

The Employee class holds attributes for name, employee ID, and basic salary, featuring methods for salary calculation and details display. The Manager class, a subclass of Employee, extends functionality by introducing department and bonus attributes and overrides the **calculateSalary()** and **displayDetails()** methods.

The program should input and display employee details and the total salary using the formula (basic salary + bonus).

**Input format :**

The first line of input consists of a string, representing the employee name.

The second line consists of an integer, representing the employee ID.

The third line consists of a double value, representing the basic salary.

The fourth line consists of a string, representing the department, it contains lowercase and uppercase letters with spaces.

The fifth line consists of a double value, representing the bonus.

**Output format :**

The output prints the ID, name, basic salary, department, bonus and total salary of the employee. Round off the total salary to two decimal places.

**Refer to the sample output for the exact text and format.**

**Code constraints :**

1000.00 ≤ basic salary ≤ 10,00,000.00

100.00 ≤ bonus ≤ 10,00,000.00

**Sample test cases :**

**Input 1 :**

Paran

1001

50000.45

Sales force

10000.37

**Output 1 :**

Employee ID: 1001

Name: Paran

Basic Salary: 50000.45

Department: Sales force

Bonus: 10000.37

Total Salary: 60000.82

**Input 2 :**

Sharon

1501

21089.75

Data Analysis

9550.15

**Output 2 :**

Employee ID: 1501

Name: Sharon

Basic Salary: 21089.75

Department: Data Analysis

Bonus: 9550.15

Total Salary: 30639.90

**Input 3 :**

John

3005

250000.70

Computer Science

102000.35

**Output 3 :**

Employee ID: 3005

Name: John

Basic Salary: 250000.7

Department: Computer Science

Bonus: 102000.35

Total Salary: 352001.05

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

String name = scanner.nextLine();

int employeeId = scanner.nextInt();

double basicSalary = scanner.nextDouble();

scanner.nextLine();

String department = scanner.nextLine();

double bonus = scanner.nextDouble();

Employee employee;

employee = new Manager(name, employeeId, basicSalary, department, bonus);

employee.displayDetails();

System.out.printf("Total Salary: %.2f\n" ,employee.calculateSalary());

}

}

class Employee {

protected String name;

protected int employeeId;

protected double basicSalary;

public Employee(String name, int employeeId, double basicSalary) {

this.name = name;

this.employeeId = employeeId;

this.basicSalary = basicSalary;

}

public double calculateSalary() {

return basicSalary;

}

public void displayDetails() {

System.out.println("Employee ID: " + employeeId);

System.out.println("Name: " + name);

System.out.println("Basic Salary: " + basicSalary);

}

}

class Manager extends Employee {

private String department;

private double bonus;

public Manager(String name, int employeeId, double basicSalary, String department, double bonus) {

super(name, employeeId, basicSalary);

this.department = department;

this.bonus = bonus;

}

public double calculateSalary() {

return basicSalary + bonus;

}

public void displayDetails() {

super.displayDetails();

System.out.println("Department: " + department);

System.out.println("Bonus: " + bonus);

}

}

Rithish is developing a straightforward pizza ordering system. To achieve this, he needs a **Pizza** class with a constructor for the base price and topping cost, along with a **calculatePrice** method overriding. He also wants a **DiscountedPizza** class that inherits from Pizza, applying a 10% discount for more than three toppings.

The program prompts the user for inputs, creates instances of both classes, calculates regular and discounted prices, and displays them formatted appropriately.

**Example 1**

**Input:**

9.5

1.25

3

**Output:**

Price without discount: Rs.13.25

Price with discount: Rs.13.25

**Explanation:**

Rithish orders a pizza with a base price of Rs. 9.5, a topping cost of Rs. 1.25, and selects 3 toppings. The price is calculated as 9.5 + (1.25 \* 3) = 13.25. The regular and discounted prices are both Rs. 13.25, as no discount has been applied.

**Example 2**

**Input:**

11.0

2.0

7

**Output:**

Price without discount: Rs.25.00

Price with discount: Rs.22.50

**Explanation:**

Rithish orders another pizza with a higher base price of Rs. 11.0, a topping cost of Rs. 2.0, and chooses 7 toppings.

Regular Price: 11.0 + (2.0 \* 7) = Rs. 25.00.

Discounted Price: The discounted price is calculated as 90% of the regular price, i.e., 0.9 \* 25.00 = Rs.22.50.

**Input format :**

The first line of input consists of a double value, representing the base price of the pizza.

The second line consists of a double value, representing the cost per topping.

The third line consists of an integer, representing the number of toppings chosen for the pizza.

**Output format :**

The first line of output prints the price without discount, rounded off to two decimal places.

The second line prints the price with the discount, rounded off to two decimal places.

**Refer to the sample output for formatting specifications.**

**Code constraints :**

The base price and the cost per topping should be greater than zero.

1 ≤ number of toppings ≤ 10

**Sample test cases :**

**Input 1 :**

9.5

1.25

3

**Output 1 :**

Price without discount: Rs.13.25

Price with discount: Rs.13.25

**Input 2 :**

11.0

2.0

7

**Output 2 :**

Price without discount: Rs.25.00

Price with discount: Rs.22.50

import java.util.Scanner;

import java.text.DecimalFormat;

public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

double basePrice = scanner.nextDouble();

double toppingCost = scanner.nextDouble();

int numberOfToppings = scanner.nextInt();

Pizza pizza = new Pizza(basePrice, toppingCost);

DiscountedPizza discountedPizza = new DiscountedPizza(basePrice, toppingCost);

DecimalFormat df = new DecimalFormat("#.00");

double regularPrice = pizza.calculatePrice(numberOfToppings);

double discountedPrice = discountedPizza.calculatePrice(numberOfToppings);

System.out.println("Price without discount: Rs." + df.format(regularPrice));

System.out.println("Price with discount: Rs." + df.format(discountedPrice));

scanner.close();

}

}

class Pizza {

private double basePrice;

private double toppingCost;

public Pizza(double basePrice, double toppingCost) {

this.basePrice = basePrice;

this.toppingCost = toppingCost;

}

public double calculatePrice(int numberOfToppings) {

return basePrice + (numberOfToppings \* toppingCost);

}

}

class DiscountedPizza extends Pizza {

public DiscountedPizza(double basePrice, double toppingCost) {

super(basePrice, toppingCost);

}

public double calculatePrice(int numberOfToppings) {

if (numberOfToppings > 3) {

return super.calculatePrice(numberOfToppings) \* 0.9;

} else {

return super.calculatePrice(numberOfToppings);

}

}

}

**Problem Statement**

Create an abstract class **Shape** with the following methods:

1. abstract void rectangleArea();
2. abstract void squareArea();
3. abstract void circleArea();

Create a class **Area** that extends **Shape** and calculates and prints all the areas. Then create a Main class, get the inputs, and pass them to the methods.

**Formula:**

1. Rectangle Area: Area = length \* breadth
2. Square Area: Area = side \* side
3. Circle Area: Area = π \* radius2

For the π value, use Math.PI from the math package.

**Input format :**

The first line of input consists of two space-separated integers, representing the length and breadth of the rectangle.

The second line consists of an integer, representing the side of the square.

The third line consists of an integer, representing the radius of the circle.

**Output format :**

The first line of output prints the integer, representing the area of a rectangle.

The second line prints the integer, representing the area of a square.

The third line prints a double value, the area of the circle, rounded off to two decimal places.

**Refer to the sample output for formatting specifications.**

**Code constraints :**

The given test cases fall under the following constraints:

1 ≤ length and breadth ≤ 100

1 ≤ side ≤ 100

1 ≤ radius ≤ 100

**Sample test cases :**

**Input 1 :**

10 20

4

5

**Output 1 :**

200

16

78.54

**Input 2 :**

15 9

12

8

**Output 2 :**

135

144

201.06

import java.io.\*;

import java.lang.Math.\*;

import java.util.\*;

import java.text.\*;

class Main {

public static void main(String [] args) {

int length,breadth,radius,side;

Scanner sc = new Scanner(System.in);

length = sc.nextInt();

breadth = sc.nextInt();

side = sc.nextInt();

radius = sc.nextInt();

Area a = new Area();

a.rectangleArea(length,breadth);

a.squareArea(side);

a.circleArea(radius);

}

}

abstract class Shape {

abstract void rectangleArea(int l,int b);

abstract void squareArea(int s);

abstract void circleArea(int r);

}

class Area extends Shape {

public void rectangleArea(int l,int b) {

System.out.println(l\*b);

}

public void squareArea(int s) {

System.out.println(s\*s);

}

public void circleArea(int r) {

DecimalFormat d = new DecimalFormat("0.00");

double res = Math.PI\*r\*r;

System.out.println(d.format(res));

}

}