**Level 1 practice problem**

**-------------------------------------------------------------------------------------------------------------------------------------**

**1. Simple Interest Calculation**

import java.util.Scanner;

public class SimpleInterestCalculator {

public static double calculateSimpleInterest(double principal, double rate, double time) {

return (principal \* rate \* time) / 100;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter Principal: ");

double principal = sc.nextDouble();

System.out.print("Enter Rate: ");

double rate = sc.nextDouble();

System.out.print("Enter Time: ");

double time = sc.nextDouble();

double si = calculateSimpleInterest(principal, rate, time);

System.out.println("The Simple Interest is " + si + " for Principal " + principal + ", Rate of Interest " + rate + " and Time " + time);

}

}

**✅ 2. Maximum Number of Handshakes**

import java.util.Scanner;

public class HandshakeCalculator {

public static int calculateHandshakes(int n) {

return (n \* (n - 1)) / 2;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter number of students: ");

int students = sc.nextInt();

int handshakes = calculateHandshakes(students);

System.out.println("Maximum number of handshakes: " + handshakes);

}

}

**✅ 3. Triangular Park Rounds (5 KM Run)**

import java.util.Scanner;

public class TriangularParkRun {

public static double calculateRounds(double a, double b, double c) {

double perimeter = a + b + c;

double distanceToRun = 5000.0; // meters

return distanceToRun / perimeter;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter side A (meters): ");

double a = sc.nextDouble();

System.out.print("Enter side B (meters): ");

double b = sc.nextDouble();

System.out.print("Enter side C (meters): ");

double c = sc.nextDouble();

double rounds = calculateRounds(a, b, c);

System.out.println("Number of rounds to complete 5 KM: " + Math.ceil(rounds));

}

}

**✅ 4. Check Positive, Negative, or Zero**

import java.util.Scanner;

public class NumberChecker {

public static int checkNumber(int num) {

if (num > 0) return 1;

else if (num < 0) return -1;

else return 0;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter a number: ");

int num = sc.nextInt();

int result = checkNumber(num);

if (result == 1) System.out.println("Positive number");

else if (result == -1) System.out.println("Negative number");

else System.out.println("Zero");

}

}

**✅ 5. Spring Season Checker**

import java.util.Scanner;

public class SpringSeason {

public static boolean isSpringSeason(int month, int day) {

return (month == 3 && day >= 20) ||

(month == 4) ||

(month == 5) ||

(month == 6 && day <= 20);

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter month (1-12): ");

int month = sc.nextInt();

System.out.print("Enter day: ");

int day = sc.nextInt();

if (isSpringSeason(month, day))

System.out.println("It's a Spring Season");

else

System.out.println("Not a Spring Season");

}

}

**✅ 6. Sum of n Natural Numbers**

import java.util.Scanner;

public class NaturalNumberSum {

public static int sumOfNaturalNumbers(int n) {

int sum = 0;

for (int i = 1; i <= n; i++) sum += i;

return sum;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter value of n: ");

int n = sc.nextInt();

int sum = sumOfNaturalNumbers(n);

System.out.println("Sum of first " + n + " natural numbers: " + sum);

}

}

**✅ 7. Smallest and Largest of 3 Numbers**

import java.util.Scanner;

public class MinMaxFinder {

public static int[] findSmallestAndLargest(int a, int b, int c) {

int min = a, max = a;

if (b < min) min = b;

if (c < min) min = c;

if (b > max) max = b;

if (c > max) max = c;

return new int[]{min, max};

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter 3 numbers: ");

int a = sc.nextInt();

int b = sc.nextInt();

int c = sc.nextInt();

int[] result = findSmallestAndLargest(a, b, c);

System.out.println("Smallest: " + result[0]);

System.out.println("Largest: " + result[1]);

}

}

**✅ 8. Quotient and Remainder**

import java.util.Scanner;

public class QuotientRemainder {

public static int[] findRemainderAndQuotient(int number, int divisor) {

int quotient = number / divisor;

int remainder = number % divisor;

return new int[]{quotient, remainder};

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter number: ");

int num = sc.nextInt();

System.out.print("Enter divisor: ");

int divisor = sc.nextInt();

int[] result = findRemainderAndQuotient(num, divisor);

System.out.println("Quotient: " + result[0]);

System.out.println("Remainder: " + result[1]);

}

}

**✅ 9. Distribute Chocolates Among Children**

import java.util.Scanner;

public class ChocolateDivider {

public static int[] findRemainderAndQuotient(int chocolates, int children) {

int each = chocolates / children;

int left = chocolates % children;

return new int[]{each, left};

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter number of chocolates: ");

int chocolates = sc.nextInt();

System.out.print("Enter number of children: ");

int children = sc.nextInt();

int[] result = findRemainderAndQuotient(chocolates, children);

System.out.println("Each child gets: " + result[0] + " chocolates");

System.out.println("Remaining chocolates: " + result[1]);

}

}

**✅ 10. Wind Chill Calculator**

import java.util.Scanner;

public class WindChillCalculator {

public static double calculateWindChill(double temp, double windSpeed) {

return 35.74 + 0.6215 \* temp + (0.4275 \* temp - 35.75) \* Math.pow(windSpeed, 0.16);

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter temperature in Fahrenheit: ");

double temp = sc.nextDouble();

System.out.print("Enter wind speed in mph: ");

double windSpeed = sc.nextDouble();

double windChill = calculateWindChill(temp, windSpeed);

System.out.printf("Wind Chill Temperature: %.2f°F\n", windChill);

}

}

**✅ 11. Trigonometric Functions (Sine, Cosine, Tangent)**

import java.util.Scanner;

public class TrigonometricFunctions {

public static double[] calculateTrigonometricFunctions(double angle) {

double radians = Math.toRadians(angle);

return new double[]{

Math.sin(radians),

Math.cos(radians),

Math.tan(radians)

};

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter angle in degrees: ");

double angle = sc.nextDouble();

double[] result = calculateTrigonometricFunctions(angle);

System.out.printf("Sine: %.4f\n", result[0]);

System.out.printf("Cosine: %.4f\n", result[1]);

System.out.printf("Tangent: %.4f\n", result[2]);

}

}

-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------

import java.util.Scanner;

public class SumOfNaturalNumbers {

// Recursive method to find sum of n natural numbers

public static int sumRecursive(int n) {

if (n == 1) {

return 1; // base case

}

return n + sumRecursive(n - 1);

}

// Formula method to find sum of n natural numbers

public static int sumFormula(int n) {

return n \* (n + 1) / 2;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter a natural number (positive integer): ");

int n = sc.nextInt();

if (n <= 0) {

System.out.println("Input is not a natural number. Please enter a positive integer.");

return;

}

int recursiveSum = sumRecursive(n);

int formulaSum = sumFormula(n);

System.out.println("Sum calculated using recursion: " + recursiveSum);

System.out.println("Sum calculated using formula: " + formulaSum);

if (recursiveSum == formulaSum) {

System.out.println("Both methods give the same result. Computation is correct!");

} else {

System.out.println("Results do not match. There might be an error.");

}

}

}

**UnitConvertor Utility Class with static methods**

public class UnitConvertor {

// Distance conversions

public static double convertKmToMiles(double km) {

double km2miles = 0.621371;

return km \* km2miles;

}

public static double convertMilesToKm(double miles) {

double miles2km = 1.60934;

return miles \* miles2km;

}

public static double convertMetersToFeet(double meters) {

double meters2feet = 3.28084;

return meters \* meters2feet;

}

public static double convertFeetToMeters(double feet) {

double feet2meters = 0.3048;

return feet \* feet2meters;

}

// Yard and Feet conversions

public static double convertYardsToFeet(double yards) {

double yards2feet = 3;

return yards \* yards2feet;

}

public static double convertFeetToYards(double feet) {

double feet2yards = 0.333333;

return feet \* feet2yards;

}

// Meters and Inches conversions

public static double convertMetersToInches(double meters) {

double meters2inches = 39.3701;

return meters \* meters2inches;

}

public static double convertInchesToMeters(double inches) {

double inches2meters = 0.0254;

return inches \* inches2meters;

}

// Inches and Centimeters conversions

public static double convertInchesToCentimeters(double inches) {

double inches2cm = 2.54;

return inches \* inches2cm;

}

// Temperature conversions

public static double convertFahrenheitToCelsius(double fahrenheit) {

return (fahrenheit - 32) \* 5 / 9;

}

public static double convertCelsiusToFahrenheit(double celsius) {

return (celsius \* 9 / 5) + 32;

}

// Weight conversions

public static double convertPoundsToKilograms(double pounds) {

double pounds2kilograms = 0.453592;

return pounds \* pounds2kilograms;

}

public static double convertKilogramsToPounds(double kilograms) {

double kilograms2pounds = 2.20462;

return kilograms \* kilograms2pounds;

}

// Volume conversions

public static double convertGallonsToLiters(double gallons) {

double gallons2liters = 3.78541;

return gallons \* gallons2liters;

}

public static double convertLitersToGallons(double liters) {

double liters2gallons = 0.264172;

return liters \* liters2gallons;

}

}

**2. StudentVoteChecker class with main to check voting eligibility of 10 students**

import java.util.Scanner;

public class StudentVoteChecker {

public boolean canStudentVote(int age) {

if (age < 0) {

return false; // negative age can't vote

}

return age >= 18;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

StudentVoteChecker checker = new StudentVoteChecker();

int[] ages = new int[10];

for (int i = 0; i < 10; i++) {

System.out.print("Enter age of student " + (i + 1) + ": ");

ages[i] = sc.nextInt();

}

System.out.println("\nVoting Eligibility Results:");

for (int i = 0; i < 10; i++) {

boolean canVote = checker.canStudentVote(ages[i]);

System.out.println("Student " + (i + 1) + " (Age: " + ages[i] + ") can vote? " + (canVote ? "Yes" : "No"));

}

}

}

---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**Java Program for Matrix Manipulation:**

import java.util.Random;

public class MatrixOperations {

// Method to create a random matrix of given rows and columns

public static double[][] createRandomMatrix(int rows, int cols) {

Random rand = new Random();

double[][] matrix = new double[rows][cols];

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

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

matrix[i][j] = rand.nextInt(10); // random integer from 0 to 9

}

}

return matrix;

}

// Method to print matrix

public static void printMatrix(double[][] matrix) {

for (double[] row : matrix) {

for (double val : row) {

System.out.printf("%8.3f ", val);

}

System.out.println();

}

}

// Method to find transpose of a matrix

public static double[][] transpose(double[][] matrix) {

int rows = matrix.length;

int cols = matrix[0].length;

double[][] transposed = new double[cols][rows];

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

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

transposed[j][i] = matrix[i][j];

}

}

return transposed;

}

import java.util.Scanner;

public class LineAndDistance {

// Method to calculate Euclidean distance

public static double euclideanDistance(double x1, double y1, double x2, double y2) {

return Math.sqrt(Math.pow(x2 - x1, 2) + Math.pow(y2 - y1, 2));

}

// Method to calculate slope and y-intercept of the line, returns array [slope, intercept]

public static double[] lineEquation(double x1, double y1, double x2, double y2) {

double m; // slope

double b; // y-intercept

if (x2 == x1) {

// Vertical line, slope is undefined

throw new ArithmeticException("Slope is undefined (vertical line).");

}

m = (y2 - y1) / (x2 - x1);

b = y1 - m \* x1;

return new double[] { m, b };

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

// Taking input for points

System.out.print("Enter x1: ");

double x1 = sc.nextDouble();

System.out.print("Enter y1: ");

double y1 = sc.nextDouble();

System.out.print("Enter x2: ");

double x2 = sc.nextDouble();

System.out.print("Enter y2: ");

double y2 = sc.nextDouble();

// Calculate distance

double distance = euclideanDistance(x1, y1, x2, y2);

System.out.printf("Euclidean distance between (%.2f, %.2f) and (%.2f, %.2f) is: %.4f\n", x1, y1, x2, y2, distance);

// Calculate line equation

try {

double[] line = lineEquation(x1, y1, x2, y2);

System.out.printf("Equation of the line: y = %.4fx + %.4f\n", line[0], line[1]);

} catch (ArithmeticException e) {

System.out.println(e.getMessage());

System.out.println("The line is vertical with equation: x = " + x1);

}

sc.close();

}

}

// Method to find determinant of a square matrix recursively

public static double determinant(double[][] matrix) {

int n = matrix.length;

if (n == 1) {

return matrix[0][0];

}

if (n == 2) {

return matrix[0][0]\*matrix[1][1] - matrix[0][1]\*matrix[1][0];

}

double det = 0;

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

det += Math.pow(-1, col) \* matrix[0][col] \* determinant(minor(matrix, 0, col));

}

return det;

}

// Method to calculate minor of a matrix (removes specified row and column)

public static double[][] minor(double[][] matrix, int row, int col) {

int n = matrix.length;

double[][] minor = new double[n - 1][n - 1];

int r = 0;

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

if (i == row) continue;

int c = 0;

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

if (j == col) continue;

minor[r][c] = matrix[i][j];

c++;

}

r++;

}

return minor;

}

// Method to find the cofactor matrix

public static double[][] cofactorMatrix(double[][] matrix) {

int n = matrix.length;

double[][] cofactor = new double[n][n];

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

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

cofactor[i][j] = Math.pow(-1, i + j) \* determinant(minor(matrix, i, j));

}

}

return cofactor;

}

// Method to multiply matrix by a scalar

public static double[][] scalarMultiply(double[][] matrix, double scalar) {

int rows = matrix.length;

int cols = matrix[0].length;

double[][] result = new double[rows][cols];

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

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

result[i][j] = matrix[i][j] \* scalar;

}

}

return result;

}

// Method to find the inverse of a matrix

public static double[][] inverse(double[][] matrix) {

double det = determinant(matrix);

if (det == 0) {

throw new ArithmeticException("Matrix is singular, can't find inverse.");

}

double[][] cofactor = cofactorMatrix(matrix);

double[][] adjoint = transpose(cofactor);

return scalarMultiply(adjoint, 1 / det);

}

// Main method to demonstrate

public static void main(String[] args) {

int rows = 3;

int cols = 3;

System.out.println("Generating random 3x3 matrix:");

double[][] matrix = createRandomMatrix(rows, cols);

printMatrix(matrix);

System.out.println("\nTranspose of the matrix:");

double[][] transposeMatrix = transpose(matrix);

printMatrix(transposeMatrix);

// Only square matrix determinant and inverse

if (rows == cols) {

System.out.println("\nDeterminant of the matrix: " + determinant(matrix));

try {

System.out.println("\nInverse of the matrix:");

double[][] inverseMatrix = inverse(matrix);

printMatrix(inverseMatrix);

} catch (ArithmeticException e) {

System.out.println(e.getMessage());

}

} else {

System.out.println("\nMatrix is not square, skipping determinant and inverse calculation.");

}

}

}