**Read a Number From Standard Input**

import java.util.Scanner;

class ReadNumber {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int num = sc.nextInt();

System.out.println("You entered: " + num);

}

}

**Get Input From the User**

import java.util.Scanner;

class GetInput {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

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

int number = sc.nextInt();

System.out.println("Number: " + number);

}

}

**Multiply Two Floating-Point Numbers**

import java.util.Scanner;

class MultiplyFloats {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

float a = sc.nextFloat();

float b = sc.nextFloat();

System.out.println("Product: " + (a \* b));

}

}

**Swap Two Numbers**

import java.util.Scanner;

class SwapNumbers {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int x = sc.nextInt(), y = sc.nextInt();

x = x + y; y = x - y; x = x - y;

System.out.println(x + " " + y);

}

}

**Add Two Binary Strings**

import java.util.Scanner;

class AddBinaryStrings {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

String b1 = sc.next(), b2 = sc.next();

System.out.println("Sum: " + Integer.toBinaryString(Integer.parseInt(b1, 2) + Integer.parseInt(b2, 2)));

}

}

**Add Two Complex Numbers**

class Complex {

int real, imag;

Complex(int r, int i) { real = r; imag = i; }

void add(Complex c) {

System.out.println("Sum: " + (real + c.real) + " + " + (imag + c.imag) + "i");

}

public static void main(String[] args) {

Complex c1 = new Complex(2, 3), c2 = new Complex(1, 4);

c1.add(c2);

}

}

**Check Even or Odd Integers**

import java.util.Scanner;

class EvenOdd {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int n = sc.nextInt();

System.out.println((n % 2 == 0) ? "Even" : "Odd");

}

}

**Find Largest Among 3 Numbers**

import java.util.Scanner;

class LargestOfThree {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int a = sc.nextInt(), b = sc.nextInt(), c = sc.nextInt();

System.out.println(Math.max(a, Math.max(b, c)));

}

}

**Find LCM of 2 Numbers**

class LCM {

static int gcd(int a, int b) { return b == 0 ? a : gcd(b, a % b); }

public static void main(String[] args) {

int a = 15, b = 20;

System.out.println((a \* b) / gcd(a, b));

}

}

**Find GCD or HCF of 2 Numbers**

class GCD {

static int gcd(int a, int b) { return b == 0 ? a : gcd(b, a % b); }

public static void main(String[] args) {

System.out.println(gcd(15, 20));

}

}

**Display All Prime Numbers from 1 to N**

import java.util.Scanner;

class PrimeNumbers {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int n = sc.nextInt();

for (int i = 2; i <= n; i++) {

if (isPrime(i)) {

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

}

}

}

static boolean isPrime(int num) {

if (num <= 1) return false;

for (int i = 2; i \* i <= num; i++) {

if (num % i == 0) return false;

}

return true;

}

}

**Check Leap Year**

import java.util.Scanner;

class LeapYear {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int year = sc.nextInt();

if ((year % 4 == 0 && year % 100 != 0) || (year % 400 == 0)) {

System.out.println("Leap Year");

} else {

System.out.println("Not a Leap Year");

}

}

}

***Check Armstrong Number Between Two Integers***

import java.util.Scanner;

class ArmstrongNumbers {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int start = sc.nextInt(), end = sc.nextInt();

for (int i = start; i <= end; i++) {

if (isArmstrong(i)) {

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

}

}

}

static boolean isArmstrong(int num) {

int sum = 0, original = num;

while (num > 0) {

int digit = num % 10;

sum += digit \* digit \* digit;

num /= 10;

}

return sum == original;

}

}

***Check if Number is Neon***

import java.util.Scanner;

class NeonNumber {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int num = sc.nextInt();

int square = num \* num, sum = 0;

while (square > 0) {

sum += square % 10;

square /= 10;

}

if (sum == num) {

System.out.println("Neon Number");

} else {

System.out.println("Not a Neon Number");

}

}

}

***Check if Input Character is Vowel or Consonant***

import java.util.Scanner;

class VowelConsonant {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

char ch = sc.next().charAt(0);

if ("AEIOUaeiou".indexOf(ch) >= 0) {

System.out.println("Vowel");

} else {

System.out.println("Consonant");

}

}

}

***Find Factorial of a Number***

import java.util.Scanner;

class Factorial {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int n = sc.nextInt(), fact = 1;

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

fact \*= i;

}

System.out.println("Factorial: " + fact);

}

}

***Find Even Sum of Fibonacci Series Till N***

import java.util.Scanner;

class FibonacciEvenSum {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int n = sc.nextInt(), a = 0, b = 1, sum = 0;

while (b <= n) {

if (b % 2 == 0) sum += b;

int temp = b;

b = a + b;

a = temp;

}

System.out.println("Even Sum: " + sum);

}

}

***Calculate Simple Interest***

import java.util.Scanner;

class SimpleInterest {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

double principal = sc.nextDouble(), rate = sc.nextDouble(), time = sc.nextDouble();

double si = (principal \* rate \* time) / 100;

System.out.println("Simple Interest: " + si);

}

}

***Calculate Compound Interest***

import java.util.Scanner;

class CompoundInterest {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

double principal = sc.nextDouble(), rate = sc.nextDouble(), time = sc.nextDouble();

double ci = principal \* Math.pow((1 + rate / 100), time) - principal;

System.out.println("Compound Interest: " + ci);

}

}

***Find the Perimeter of a Rectangle***

import java.util.Scanner;

class RectanglePerimeter {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

double length = sc.nextDouble(), width = sc.nextDouble();

System.out.println("Perimeter: " + 2 \* (length + width));

}

}

***BFS***

import java.util.Scanner;

public class Main {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        int node, edge;

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

        node = sc.nextInt();

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

        edge = sc.nextInt();

        // Create an adjacency matrix for the graph

        int graph[][] = new int[node][node];

        System.out.println("Enter the " + edge + " edges (u v):");

        int u, v;

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

            u = sc.nextInt();

            v = sc.nextInt();

            graph[u][v] = graph[v][u] = 1; // Undirected graph, so mark both ways

        }

        int visit[] = new int[node]; // To keep track of visited nodes

        int distance[] = new int[node]; // To store the distance from the source

        int parent[] = new int[node]; // To store the parent of each node

        int src = 1; // Set the source node (can be changed)

        System.out.print("Enter source node: ");

        src = sc.nextInt();

        System.out.print("Enter destination node: ");

        int dest = sc.nextInt();

        // Initialize the BFS traversal

        enq(src);

        visit[src] = 1; // Mark the source as visited

        distance[src] = 0; // Distance from source to itself is 0

        parent[src] = -1; // Source has no parent

        // BFS Loop

        System.out.println("The BFS traversal:");

        while (front < rear) {

            int x = deq();

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

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

                if (graph[x][i] == 1 && visit[i] == 0) { // If there is an edge and i is unvisited

                    visit[i] = 1; // Mark as visited

                    distance[i] = distance[x] + 1; // Update distance

                    parent[i] = x; // Set parent for path reconstruction

                    enq(i);

                }

            }

            visit[x] = 2; // Mark node as fully processed

        }

        System.out.println("\nShortest distance from " + src + " to " + dest + " is: " + distance[dest]);

        printPath(src, dest, parent);

        sc.close();

    }

    // Queue Implementation

    static int queue[] = new int[100];

    static int front = -1, rear = -1;

    static void enq(int n) {

        if (rear == -1) {

            front = 0;

        }

        queue[++rear] = n;

    }

    static int deq() {

        return queue[front++];

    }

    // Function to print the shortest path

    static void printPath(int src, int dest, int[] parent) {

        if (dest == -1) {

            System.out.println("No path exists.");

            return;

        }

        // Reconstruct the path by following parent links

        if (src == dest) {

            System.out.print(src);

        } else if (parent[dest] == -1) {

            System.out.println("No path exists from " + src + " to " + dest);

        } else {

            printPath(src, parent[dest], parent);

            System.out.print(" -> " + dest);

        }

    }

}

**DFS**

import java.util.\*;

public class Main {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        int node, edge;

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

        node = sc.nextInt();

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

        edge = sc.nextInt();

        int[][] graph = new int[node][node];

        System.out.println("Enter the " + edge + " edges (u v):");

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

            int u = sc.nextInt();

            int v = sc.nextInt();

                graph[u][v] = 1;

            }

        System.out.println("Topological Sort:");

        topologicalSort(graph, node);

    }

    static void topologicalSort(int[][] graph, int node) {

        boolean[] visited = new boolean[node];

        int[] result = new int[node];

        int[] index = {node - 1};

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

            if (!visited[i]) {

                dfs(graph, i, visited, result, index);

            }

        }

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

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

        }

        System.out.println();

    }

    static void dfs(int[][] graph, int v, boolean visited[], int result[], int index[]) {

        visited[v] = true;

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

            if (graph[v][i] == 1 && !visited[i]) {

                dfs(graph, i, visited, result, index);

            }

        }

        result[index[0]] = v;

        index[0]--;

    }

}