**Data Structures**

1. Implement a function that receives an array of integers and returns the first repeating element. If there are no repeating elements, return -1.

**Input Format**

* The input begins with an integer N (the size of the array), followed by N space-separated integers (the elements of the array).

**Output Format**

* Output the first repeating element.
* Output -1 if no such element exists.
* If the input is non-integer print **Invalid input**.

**Solution**

import java.util.\*;

public class FirstRepeatingElement {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

try {

int n = Integer.parseInt(scanner.next());

if (n < 1 || n > 1000) {

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

return;

}

int[] arr = new int[n];

Set<Integer> seen = new HashSet<>();

Set<Integer> repeated = new HashSet<>();

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

if (!scanner.hasNextInt()) {

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

return;

}

arr[i] = scanner.nextInt();

if (seen.contains(arr[i])) {

repeated.add(arr[i]);

} else {

seen.add(arr[i]);

}

}

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

if (repeated.contains(arr[i])) {

System.out.println(arr[i]);

return;

}

}

System.out.println("-1");

} catch (Exception e) {

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

} finally {

scanner.close();

}

}

}

**Sample Input**

5

1 2 3 2 5

**Output**

2

2. Given an integer array, implement a function that moves all zeroes in the array to the end while maintaining the relative order of the non-zero elements. The function should modify the array in place.

**Input Format**

* An integer n represents the number of elements in the array.
* An array arr of length n containing integer values representing the elements of the array.

**Output Format**

* Print the modified array with all zeroes moved to the end.

**Solution**

import java.util.\*;

public class MoveZeroesToEnd {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int n = scanner.nextInt();

int[] arr = new int[n];

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

arr[i] = scanner.nextInt();

}

moveZeroesToEnd(arr);

for (int num : arr) {

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

}

System.out.println();

scanner.close();

}

public static void moveZeroesToEnd(int[] arr) {

int index = 0;

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

if (arr[i] != 0) {

int temp = arr[index];

arr[index++] = arr[i];

arr[i] = temp;

}

}

}

}

**Sample Input**

6

0 1 0 3 12 0

**Output**

1 3 12 0 0 0

3. Given an array, return the duplicates present in the array.

**Input Format**

* An integer n represents the number of elements in the array.
* A list of n integers, indicating the elements of the array.

**Output Format**

* If duplicates are found, print each duplicate on a new line in the order they appear.
* If no duplicates are found, print **No duplicates found**.

**Solution**

import java.util.Scanner;

public class FindAllDuplicates {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

int n = scanner.nextInt();

if (n <= 0) {

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

return;

}

int[] array = new int[n];

// System.out.println("Enter the elements (positive integers only):");

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

int value = scanner.nextInt();

if (value < 0) {

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

return;

}

array[i] = value;

}

// System.out.println("Duplicates:");

findDuplicates(array);

scanner.close();

}

private static void findDuplicates(int[] array) {

boolean hasDuplicates = false;

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

for (int j = i + 1; j < array.length; j++) {

if (array[i] == array[j]) {

System.out.println(array[i]);

hasDuplicates = true;

break;

}

}

}

if (!hasDuplicates) {

System.out.println("No duplicates found");

}

}

}

**Input Format**

6

3 5 2 3 7 5

**Output Format**

3

5

4. Given an array of integers, implement a function that doubles the values at all odd indices while keeping the values at even indices unchanged.

**Input Format**

* The first line of input contains an integer n representing the number of elements in the array.
* The second line of input contains n integers representing the elements of the array.

**Output Format**

* Print the array after doubling the values of elements at odd indices.

**Solution**

import java.util.Scanner;

public class DoubleOddIndices {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int n = scanner.nextInt();

int[] arr = new int[n];

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

arr[i] = scanner.nextInt();

}

doubleOddIndices(arr);

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

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

}

System.out.println();

scanner.close();

}

public static void doubleOddIndices(int[] arr) {

for (int i = 1; i < arr.length; i += 2) {

arr[i] \*= 2;

}

}

}

**Sample Input**

5

1 2 3 4 5

**Output**

1 4 3 8 5

5. Given a matrix of size m × n, implement a function that reads the matrix elements from the user and prints its transpose.

**Input Format**

* An integer m number of rows.
* An integer n number of columns.
* m×n integers representing the matrix values.

**Output Format**

* Print the transposed matrix with values.
* If the input is invalid (e.g., not integers or dimensions out of bounds), print Invalid input.

**Solution**

import java.util.Scanner;

class Matrix {

private int[][] matrix;

private int m, n;

public Matrix(int m, int n) {

this.m = m;

this.n = n;

matrix = new int[m][n];

}

public void setValues(Scanner scanner) {

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

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

if (scanner.hasNextInt()) {

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

} else {

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

System.exit(0); // Terminate the program on invalid input

}

}

}

}

public String getTranspose() {

StringBuilder sb = new StringBuilder();

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

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

sb.append(matrix[i][j]);

if (i < m - 1) sb.append(" "); // Append space between elements in a row

}

if (j < n - 1) sb.append("\n"); // Append newline after each row except the last one

}

return sb.toString();

}

}

public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Validate that the input is an integer and within the correct range

if (scanner.hasNextInt()) {

int m = scanner.nextInt();

if (scanner.hasNextInt()) {

int n = scanner.nextInt();

// Validate that m and n are within the valid range (1 to 100)

if (m >= 1 && m <= 100 && n >= 1 && n <= 100) {

Matrix matrix = new Matrix(m, n);

matrix.setValues(scanner);

System.out.println(matrix.getTranspose());

} else {

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

}

} else {

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

}

} else {

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

}

scanner.close();

}

}

**Sample Input**

2 3

1 2 3

4 5 6

**Output**

1 4

2 5

3 6

6. Given a singly linked list of **n** integers, implement a function that sorts the linked list using the insertion sort algorithm.

**Input Format**

* The first line contains an integer n, the number of elements in the linked list.
* The second line contains n integers, representing the elements of the linked list.

**Output Format**

* Print the elements of the sorted linked list as space-separated integers.

**Solution**

import java.util.Scanner;

class ListNode {

int val;

ListNode next;

ListNode(int val) {

this.val = val;

this.next = null;

}

}

public class InsertionSortLinkedList {

public static ListNode insertionSortList(ListNode head) {

if (head == null) return null;

ListNode sorted = null;

while (head != null) {

ListNode current = head;

head = head.next;

if (sorted == null || sorted.val >= current.val) {

current.next = sorted;

sorted = current;

} else {

ListNode temp = sorted;

while (temp.next != null && temp.next.val < current.val) {

temp = temp.next;

}

current.next = temp.next;

temp.next = current;

}

}

return sorted;

}

public static void printList(ListNode head) {

while (head != null) {

System.out.print(head.val + " ");

head = head.next;

}

System.out.println();

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// System.out.print("Enter the number of elements in the linked list: ");

int n = scanner.nextInt();

if (n < 0) {

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

return;

}

ListNode head = null;

ListNode tail = null;

// System.out.print("Enter the elements of the linked list: ");

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

int value = scanner.nextInt();

ListNode newNode = new ListNode(value);

if (head == null) {

head = newNode;

tail = newNode;

} else {

tail.next = newNode;

tail = newNode;

}

}

head = insertionSortList(head);

printList(head);

scanner.close();

}

}

**Sample Input**

5

40 20 10 30 50

**Output**

10 20 30 40 50

7. You are given a singly linked list and an integer k. Your task is to reverse every k consecutive nodes in the linked list. If there are fewer than k nodes left at the end of the list, leave them as they are. The process should continue throughout the entire list, and the modified list should be printed.

**Input Format**

* A list of integers represents the value of the nodes.
* Second line contains an integer k.

**Output Format**

* Print the modified list after reversing every k nodes.

**Solution**

import java.util.Scanner;

public class ReverseKGroup {

// Definition for singly-linked list.

static class ListNode {

int val;

ListNode next;

ListNode(int val) { this.val = val; }

}

// Function to reverse k nodes in the linked list

public static ListNode reverseKGroup(ListNode head, int k) {

if (k <= 0) return head;

ListNode dummy = new ListNode(0);

dummy.next = head;

ListNode prevGroupEnd = dummy;

ListNode current = head;

while (current != null) {

ListNode groupStart = current;

ListNode groupEnd = current;

// Check if there are k nodes left in the list

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

groupEnd = groupEnd.next;

if (groupEnd == null) return dummy.next;

}

ListNode nextGroupStart = groupEnd.next;

ListNode prev = null;

ListNode curr = groupStart;

// Reverse k nodes

while (curr != nextGroupStart) {

ListNode next = curr.next;

curr.next = prev;

prev = curr;

curr = next;

}

// Connect reversed group with previous and next groups

prevGroupEnd.next = prev;

groupStart.next = nextGroupStart;

prevGroupEnd = groupStart;

current = nextGroupStart;

}

return dummy.next;

}

// Function to print the linked list

public static void printList(ListNode head) {

ListNode current = head;

while (current != null) {

System.out.print(current.val + " ");

current = current.next;

}

System.out.println();

}

// Function to create a linked list from user input

public static ListNode createListFromInput(String input) {

String[] parts = input.trim().split("\\s+");

if (parts.length == 0) return null;

ListNode head = new ListNode(Integer.parseInt(parts[0]));

ListNode current = head;

for (int i = 1; i < parts.length; i++) {

try {

ListNode newNode = new ListNode(Integer.parseInt(parts[i]));

current.next = newNode;

current = newNode;

} catch (NumberFormatException e) {

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

return null;

}

}

return head;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// System.out.println("Enter the linked list elements separated by spaces:");

String listInput = scanner.nextLine();

// System.out.println("Enter the integer k:");

int k;

try {

k = Integer.parseInt(scanner.nextLine().trim());

} catch (NumberFormatException e) {

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

return;

}

// Create the linked list from the input

ListNode head = createListFromInput(listInput);

if (head == null) return;

// Reverse the nodes in k groups

ListNode newHead = reverseKGroup(head, k);

// Print the modified linked list

// System.out.println("Modified linked list:");

printList(newHead);

scanner.close();

}

}

**Sample Input**

1 2 3 4 5 6 7 8 9

4

**Output**

4 3 2 1 8 7 6 5 9

8. Given an n × n square matrix, implement a function to rotate the matrix 90 degrees clockwise.

**Input Format**

* The first line contains an integer n representing the size of the matrix.
* The next **n** lines each contain n integers, representing the elements of each row in the matrix.

**Output Format**

* Print the matrix after rotating it by 90 degrees clockwise. Each row of the rotated matrix should be printed on a new line.

**Solution**

import java.util.Scanner;

public class MatrixRotation {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int n = scanner.nextInt();

if (n < 1 || n > 500) {

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

return;

}

int[][] matrix = new int[n][n];

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

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

if (scanner.hasNextInt()) {

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

} else {

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

return;

}

}

}

// Rotate the matrix by 90 degrees clockwise

rotateMatrix(matrix, n);

// Print the rotated matrix

printMatrix(matrix, n);

}

private static void rotateMatrix(int[][] matrix, int n) {

// Transpose the matrix

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

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

int temp = matrix[i][j];

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

matrix[j][i] = temp;

}

}

// Reverse each row

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

for (int j = 0; j < n / 2; j++) {

int temp = matrix[i][j];

matrix[i][j] = matrix[i][n - 1 - j];

matrix[i][n - 1 - j] = temp;

}

}

}

private static void printMatrix(int[][] matrix, int n) {

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

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

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

}

System.out.println();

}

}

}

**Sample Input**

2

5 4

1 5

**Output**

1 5

5 4

9. Given an integer n representing the number of nodes in a binary tree, followed by n integers representing the node values in level-order insertion, implement a function to construct the binary tree and determine its height.

**Input Format**

* The first line contains an integer n, the number of nodes in the binary tree.
* The next n lines contain integers representing the keys of the nodes, inserted level-wise (starting from the root).

**Output Format**

* Print the height of the binary tree.

**Solution**

import java.util.LinkedList;

import java.util.Queue;

import java.util.Scanner;

class BinaryTree {

static class Node {

int data;

Node left, right;

public Node(int item) {

data = item;

left = right = null;

}

}

Node root;

// Insert nodes level-wise

public void insertLevelOrder(int[] keys) {

Queue<Node> queue = new LinkedList<>();

root = new Node(keys[0]);

queue.add(root);

int i = 1;

while (!queue.isEmpty() && i < keys.length) {

Node current = queue.poll();

if (i < keys.length) {

current.left = new Node(keys[i++]);

queue.add(current.left);

}

if (i < keys.length) {

current.right = new Node(keys[i++]);

queue.add(current.right);

}

}

}

// Calculate the height of the binary tree

public int calculateHeight(Node root) {

if (root == null) return -1; // Height of an empty tree is -1

int leftHeight = calculateHeight(root.left);

int rightHeight = calculateHeight(root.right);

return 1 + Math.max(leftHeight, rightHeight);

}

// Handle invalid input

public static boolean validateInput(int n, int[] keys) {

if (n < 0 || n > 10) {

return false;

}

for (int key : keys) {

if (key < -100 || key > 100) {

return false;

}

}

return true;

}

}

public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int n = scanner.nextInt();

scanner.nextLine(); // Consume newline

int[] keys = new int[n];

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

try {

keys[i] = Integer.parseInt(scanner.nextLine());

} catch (NumberFormatException e) {

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

return;

}

}

if (!BinaryTree.validateInput(n, keys)) {

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

return;

}

BinaryTree tree = new BinaryTree();

if (n > 0) {

tree.insertLevelOrder(keys);

System.out.println(tree.calculateHeight(tree.root));

} else {

System.out.println(-1); // Empty tree height

}

scanner.close();

}

}

**Sample Input**

3

10

20

30

**Output**

1

10. Given an integer n representing the number of nodes in a binary search tree (BST), followed by n integers representing the node values in level-order insertion, and an additional integer **searchKey**, implement a function to construct the BST and determine whether the **searchKey** exists in the tree.

**Input Format**

* The first line contains an integer n, the number of nodes in the binary search tree.
* The next n lines contain integers representing the keys of the nodes, inserted level-wise.
* The last line contains the integer value to search in the BST.

**Output Format**

* Print **Found** if the value exists in the tree, otherwise print **Not Found**.

**Sample Input**

5

10

5

15

2

7

5

**Output**

Found

11. Given a 4 × 4 adjacency matrix representing a directed graph, implement a function to detect if the graph contains a cycle using Depth First Search (DFS) with recursion.

**Input Format**

* The input starts with a V×V matrix, where V=4. Each of the next 4 lines contains 4 integers representing the adjacency matrix of the graph. All integers will be either 0 or 1.

**Output Format**

* Print **Graph contains cycle** if the graph contains a cycle.
* Print **Graph doesn't contain cycle** if no cycle exists.

**Solution**

import java.util.Scanner;

public class GraphCycleDetection {

static final int V = 4;

static boolean isCyclicUtil(int v, boolean[] visited, boolean[] recStack, int[][] graph) {

if (!visited[v]) {

visited[v] = true;

recStack[v] = true;

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

if (graph[v][i] != 0) { // If there's an edge

if (!visited[i] && isCyclicUtil(i, visited, recStack, graph))

return true;

else if (recStack[i])

return true;

}

}

}

recStack[v] = false; // Remove the vertex from recursion stack

return false;

}

static boolean isCyclic(int[][] graph) {

boolean[] visited = new boolean[V];

boolean[] recStack = new boolean[V];

// Check all vertices for cycle detection

for (int i = 0; i < V; i++)

if (isCyclicUtil(i, visited, recStack, graph))

return true;

return false;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int[][] graph = new int[V][V];

boolean isValidInput = true;

// System.out.println("Enter the adjacency matrix values (0 or 1):");

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

// System.out.println("Enter values for row " + (i + 1) + ": ");

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

if (scanner.hasNextInt()) {

int input = scanner.nextInt();

// Check if input is 0 or 1

if (input != 0 && input != 1) {

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

isValidInput = false;

break;

}

graph[i][j] = input;

} else {

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

isValidInput = false;

break;

}

}

if (!isValidInput) break;

}

// If the input is valid, check for a cycle

if (isValidInput) {

if (isCyclic(graph)) {

System.out.println("Graph contains cycle");

} else {

System.out.println("Graph doesn't contain cycle");

}

}

scanner.close();

}

}

**Sample Input**

0 1 0 0

0 0 1 0

0 0 0 1

1 0 0 0

**Output**

Graph contains cycle

12. Given an array of integers height representing the heights of vertical lines on a 2D plane, where each line's width is 1, find the maximum area of water that can be trapped between any two lines. The area is determined by the width of the container and the height of the shorter line.

**Input Format**

* An integer n, is the number of elements in the array height.
* An array of n integers, where each height[i] represents the height of a vertical line.

**Output Format**

* A single integer represents the maximum area of water that can be trapped between two lines.

**Solution**

import java.util.Scanner;

class Solution {

public int maxArea(int[] height) {

int left = 0;

int right = height.length - 1;

int res = 0;

while (left < right) {

int containerLength = right - left;

int area = containerLength \* Math.min(height[left], height[right]);

res = Math.max(res, area);

if (height[left] < height[right]) {

left++;

} else {

right--;

}

}

return res;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int n = -1;

boolean invalidInput = false;

while (n <= 0) {

if (invalidInput) {

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

invalidInput = false;

}

// System.out.print("Enter the number of elements (positive integer): ");

if (scanner.hasNextInt()) {

n = scanner.nextInt();

if (n <= 0) {

invalidInput = true;

n = -1;

}

} else {

invalidInput = true;

scanner.next();

}

}

int[] height = new int[n];

int i = 0;

invalidInput = false;

while (i < n) {

if (invalidInput) {

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

invalidInput = false;

}

// System.out.print("Enter element " + (i + 1) + ": ");

if (scanner.hasNextInt()) {

height[i] = scanner.nextInt();

i++;

} else {

invalidInput = true;

scanner.next();

}

}

Solution solution = new Solution();

int result = solution.maxArea(height);

System.out.println(result);

scanner.close();

}

}

**Sample Input**

8

7 1 2 3 9 4 5 6

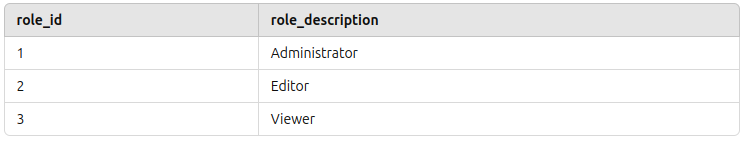
**Output**

42

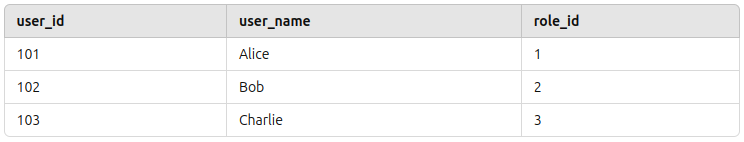
**Database Management System**

13. In a content management system (CMS), users are assigned different roles based on their access level. The system has three types of roles: Administrator, Editor, and Viewer. Each user is linked to a specific role, which defines their permissions in the system. Your task to retrieve the names of all users along with their role descriptions. Use an INNER JOIN to combine data from the users and roles tables.

**Input Table**



**Users Table**



**Solution**

CREATE TABLE roles (

role\_id INT NOT NULL PRIMARY KEY,

role\_description VARCHAR(100) NOT NULL

);

CREATE TABLE users (

user\_id INT NOT NULL PRIMARY KEY,

user\_name VARCHAR(100) NOT NULL,

role\_id INT NOT NULL,

FOREIGN KEY (role\_id) REFERENCES roles(role\_id)

);

INSERT INTO roles (role\_id, role\_description)

VALUES

(1, 'Administrator'),

(2, 'Editor'),

(3, 'Viewer');

INSERT INTO users (user\_id, user\_name, role\_id)

VALUES

(101, 'Alice', 1),

(102, 'Bob', 2),

(103, 'Charlie', 3);

SELECT

u.user\_name,

r.role\_description

FROM

users u

INNER JOIN

roles r

ON

u.role\_id = r.role\_id;

**Output Table**

**user\_name | role\_description**

**--------- | ----------------**

**Alice | Administrator**

**Bob | Editor**

**Charlie | Viewer**

14. Emma is a Human Resources (HR) manager at a company. She wants to maintain a record of all employees in a table called Employees. She needs to add a new employee, John Doe, who has just joined the HR department.

The Employees table has the following columns:

* ID (Unique Employee ID)
* Name (Employee's Full Name)
* Age (Employee's Age)
* Department (Department Name)

Implement an SQL query to insert John Doe’s details (ID = 101, Name = 'John Doe', Age = 30, Department = 'HR')

**Solution**

-- Create the Employees table

CREATE TABLE Employees (

ID INT PRIMARY KEY,

Name VARCHAR(100),

Age INT,

Department VARCHAR(50)

);

-- Insert a new employee

INSERT INTO Employees (ID, Name, Age, Department)

VALUES (101, 'John Doe', 30, 'HR');

Select \* from Employees;

**Output Table**

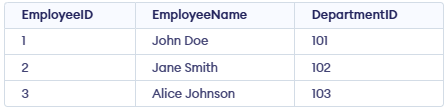
**ID | Name | Age | Department**

**--- | -------- | --- | ----------**

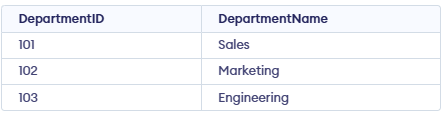
**101 | John Doe | 30 | HR**

15. Implement a SQL query to retrieve a list of all employees along with their department names.You have two tables:

* Employees with columns: EmployeeID, EmployeeName, DepartmentID.



* Departments with columns: DepartmentID, DepartmentName.



**Solution**

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY,

EmployeeName VARCHAR(100),

DepartmentID INT

);

CREATE TABLE Departments (

DepartmentID INT PRIMARY KEY,

DepartmentName VARCHAR(100)

);

INSERT INTO Employees (EmployeeID, EmployeeName, DepartmentID)

VALUES

(1, 'John Doe', 101),

(2, 'Jane Smith', 102),

(3, 'Alice Johnson', 103);

INSERT INTO Departments (DepartmentID, DepartmentName)

VALUES

(101, 'Sales'),

(102, 'Marketing'),

(103, 'Engineering');

SELECT E.EmployeeID, E.EmployeeName, D.DepartmentName

FROM Employees E

JOIN Departments D ON E.DepartmentID = D.DepartmentID;

**Output**

**EmployeeID | EmployeeName | DepartmentName**

**---------- | ------------- | --------------**

**1 | John Doe | Sales**

**2 | Jane Smith | Marketing**

**3 | Alice Johnson | Engineering**

**OOPS**

16. A job recruitment portal allows companies to post job openings for full-time jobs and internships. You need to implement a Job class with attributes jobTitle and companyName. This class should be extended into:

* FullTimeJob (inherits Job)
* Internship (inherits Job and includes an additional attribute, stipend)

Each class should include a method to display job details appropriately.

**Input Format**

* An integer N representing the number of job postings.
* The next N lines contain:
* Job Type (either "FullTime" or "Internship").
  + A string jobTitle
  + A string companyName
  + If the job is an Internship, a floating-point stipend

**Output Format**

* If the input is valid, print:
  + For Full-Time Job: **Full-Time Job: <jobTitle> at <companyName>**
  + For Internship: Internship: **<jobTitle> at <companyName> with Stipend <stipend>**

**Solution**

import java.util.\*;

class Job {

protected String jobTitle;

protected String companyName;

public Job(String jobTitle, String companyName) {

this.jobTitle = jobTitle;

this.companyName = companyName;

}

public void display() {

System.out.println("Job: " + jobTitle + " at " + companyName);

}

}

class FullTimeJob extends Job {

public FullTimeJob(String jobTitle, String companyName) {

super(jobTitle, companyName);

}

@Override

public void display() {

System.out.println("Full-Time Job: " + jobTitle + " at " + companyName);

}

}

class Internship extends Job {

private double stipend;

public Internship(String jobTitle, String companyName, double stipend) {

super(jobTitle, companyName);

this.stipend = stipend;

}

@Override

public void display() {

System.out.println("Internship: " + jobTitle + " at " + companyName + " with Stipend " + stipend);

}

}

public class JobRecruitmentPortal {

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++) {

String[] input = sc.nextLine().split(" ");

if (input.length < 3) {

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

continue;

}

String jobType = input[0];

String jobTitle = input[1];

String companyName = input[2];

// Validate jobTitle and companyName length

if (jobTitle.length() < 1 || jobTitle.length() > 50 || companyName.length() < 1 || companyName.length() > 50) {

System.out.println("Invalid");

continue;

}

if (jobType.equals("FullTime")) {

FullTimeJob job = new FullTimeJob(jobTitle, companyName);

job.display();

} else if (jobType.equals("Internship")) {

if (input.length != 4) {

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

continue;

}

try {

double stipend = Double.parseDouble(input[3]);

if (stipend < 1000 || stipend > 50000) {

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

continue;

}

Internship internship = new Internship(jobTitle, companyName, stipend);

internship.display();

} catch (NumberFormatException e) {

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

}

} else {

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

}

}

sc.close();

}

}

**Sample Input**

1

FullTime Developer Google

**Output**

Full-Time Job: Developer at Google

17. You are working on a vehicle tracking system, implementing a base class Vehicle with attributes and methods for general vehicle information. Create a subclass ElectricVehicle that extends Vehicle and includes additional attributes and methods specific to electric vehicles.

**Input Format**

1. For Vehicle

A single line with the make and model: make model

2. For ElectricVehicle

A single line with the make, model, battery capacity (in kWh), and charging status: make model battery\_capacity charging\_status (True or False)

**Output Format**

* Display the vehicle make and model.

**Solution**

import java.util.Scanner;

class Vehicle {

private String make;

private String model;

public Vehicle(String make, String model) {

if (make.isEmpty() || model.isEmpty()) {

throw new IllegalArgumentException("Make and model must be non-empty.");

}

this.make = make;

this.model = model;

}

public void displayInfo() {

System.out.println("" + make);

System.out.println("" + model);

}

}

class ElectricVehicle extends Vehicle {

private double batteryCapacity; // in kWh

private boolean chargingStatus; // true if charging, false if not

public ElectricVehicle(String make, String model, double batteryCapacity, boolean chargingStatus) {

super(make, model);

if (batteryCapacity <= 0) {

throw new IllegalArgumentException("Battery capacity must be a positive number.");

}

this.batteryCapacity = batteryCapacity;

this.chargingStatus = chargingStatus;

}

@Override

public void displayInfo() {

super.displayInfo();

System.out.println("" + batteryCapacity + " kWh");

System.out.println("" + (chargingStatus ? "Charging" : "Not Charging"));

}

}

public class VehicleTrackingSystem {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// System.out.println("Enter vehicle make and model:");

String vehicleInput = scanner.nextLine();

String[] vehicleData = vehicleInput.split(" ");

Vehicle vehicle = new Vehicle(vehicleData[0], vehicleData[1]);

vehicle.displayInfo();

// System.out.println("\nEnter electric vehicle make, model, battery capacity, and charging status:");

String electricVehicleInput = scanner.nextLine();

String[] electricVehicleData = electricVehicleInput.split(" ");

String make = electricVehicleData[0];

String model = electricVehicleData[1];

double batteryCapacity = Double.parseDouble(electricVehicleData[2]);

boolean chargingStatus = Boolean.parseBoolean(electricVehicleData[3]);

ElectricVehicle electricVehicle = new ElectricVehicle(make, model, batteryCapacity, chargingStatus);

electricVehicle.displayInfo();

scanner.close();

}

}

**Sample Input**

Toyota Camry

Tesla ModelS 75 true

**Output**

Toyota

Camry

Tesla

ModelS

75.0 kWh

Charging

18. You are given the dimensions of a rectangle, specifically its length and width. Your task is to calculate and print the area and perimeter of the rectangle.

**Input Format**

* The first line of input contains a single floating-point number length, representing the length of the rectangle.
* The second line of input contains a single floating-point number width, representing the width of the rectangle.

**Output Format**

* Print the area of the rectangle on the first line and the perimeter of the rectangle on the second line single floating-point number.

**Code**

import java.util.Scanner;

class Rectangle {

public double length;

public double width;

public double calculateArea() {

return length \* width;

}

public double calculatePerimeter() {

return 2 \* (length + width);

}

public void displayResults() {

System.out.println(calculateArea());

System.out.println(calculatePerimeter());

}

}

public class RectangleMain {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

Rectangle rectangle = new Rectangle();

// System.out.print("Enter the length of the rectangle: ");

rectangle.length = scanner.nextDouble();

// System.out.print("Enter the width of the rectangle: ");

rectangle.width = scanner.nextDouble();

if (rectangle.length <= 0 || rectangle.width <= 0) {

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

} else {

rectangle.displayResults();

}

}

}

**Sample Input**

5.5

3.0

**Output**

16.5

17.0

19. You are given the lengths of three sides of a triangle. Create a class Triangle to determine if these sides form a valid triangle(based on triangular inequality: a + b > c) and, if so, classify the type of triangle they form. A triangle can be classified as:

* Equilateral: All three sides are equal.
* Isosceles: Exactly two sides are equal.
* Scalene: All three sides are different.

**Input Format**

* Three floating-point numbers representing the lengths of the sides of the triangle (side1, side2, side3).

**Output Format**

* If the input values form a valid triangle, the program should print the type of triangle equilateral, isosceles, or scalene.

**Code**

import java.util.Scanner;

class Triangle {

private double side1;

private double side2;

private double side3;

public Triangle(double s1, double s2, double s3) {

side1 = s1;

side2 = s2;

side3 = s3;

}

public boolean isEquilateral() {

return (side1 == side2 && side2 == side3);

}

public boolean isIsosceles() {

return (side1 == side2 || side1 == side3 || side2 == side3);

}

public boolean isScalene() {

return (!isEquilateral() && !isIsosceles());

}

public boolean isValidTriangle() {

return (side1 + side2 > side3 && side1 + side3 > side2 && side2 + side3 > side1);

}

}

public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

double s1 = scanner.nextDouble();

double s2 = scanner.nextDouble();

double s3 = scanner.nextDouble();

Triangle triangle = new Triangle(s1, s2, s3);

if (triangle.isValidTriangle()) {

if (triangle.isEquilateral()) {

System.out.println("equilateral");

} else if (triangle.isIsosceles()) {

System.out.println("isosceles");

} else if (triangle.isScalene()) {

System.out.println("scalene");

}

} else {

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

}

scanner.close();

}

}

**Sample Input**

3.0 3.0 3.0

**Output**

Equilateral

20. Dharani is planning to apply for a loan. She can choose between a home loan and a personal loan. Each loan type has a specific interest rate of 5% for a home loan and 10% for a personal loan. Samantha wants to enter the loan amount and type and get an instant interest calculation. Implement an abstract class to help Dharani calculate the loan interest amount based on her input.

**Input Format**

* A double value for loanAmount represents the loan amount Dharani wants.
* A string value for loanType represents the type of loan Dharani chooses.

**Output Format**

* Print the interest for the specified loan.

**Code**

import java.util.Scanner;

abstract class Loan {

protected double loanAmount;

public Loan(double loanAmount) {

this.loanAmount = loanAmount;

}

public abstract double calculateInterest();

public void displayLoanDetails() {

System.out.print( calculateInterest());

}

}

class HomeLoan extends Loan {

private static final double INTEREST\_RATE = 0.05;

public HomeLoan(double loanAmount) {

super(loanAmount);

}

@Override

public double calculateInterest() {

return loanAmount \* INTEREST\_RATE;

}

}

class PersonalLoan extends Loan {

private static final double INTEREST\_RATE = 0.10;

public PersonalLoan(double loanAmount) {

super(loanAmount);

}

@Override

public double calculateInterest() {

return loanAmount \* INTEREST\_RATE;

}

}

public class LoanCalculator {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

try {

double loanAmount = scanner.nextDouble();

if (loanAmount <= 0) throw new IllegalArgumentException("Invalid Input");

String loanType = scanner.next();

Loan loan;

if (loanType.equalsIgnoreCase("HomeLoan")) {

loan = new HomeLoan(loanAmount);

} else if (loanType.equalsIgnoreCase("PersonalLoan")) {

loan = new PersonalLoan(loanAmount);

} else {

throw new IllegalArgumentException("Invalid Input");

}

loan.displayLoanDetails();

} catch (Exception e) {

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

} finally {

scanner.close();

}

}

}

**Sample Input**

5000

HomeLoan

**Output**

250.0