**Scenario-Based Questions – Java Programming**

1.Write a Java program named AllEqual.java that takes three integer command-line arguments and prints "equal" if all three integers are equal, and "not equal" otherwise. Ensure the program handles edge cases such as non-integer inputs and the incorrect number of arguments gracefully.

Sample Input & Output:

java AllEqual 4 4 4 equal

java AllEqual 4 5 6 not equal

Test Cases:

Input: java AllEqual 3 3 3 Output: equal

Input: java AllEqual 3 3 4 Output:not equal

Input: java AllEqual 1 1 1 Output:Invalid input

Input: java AllEqual 2.5 3 4 Output:Invalid input

Input: java AllEqual abc 2 3 Output:Invalid input

Input: java AllEqual 3

Output: Please provide exactly three integer arguments.

PROGRAMM:-

public class AllEqual {

public static void main(String[] args) {

// Check if the correct number of arguments is provided

if (args.length != 3) {

System.out.println("Error: Exactly three integer arguments are required.");

return;

}

int[] numbers = new int[3];

// Attempt to parse each argument as an integer

for (int i = 0; i < 3; i++) {

try {

numbers[i] = Integer.parseInt(args[i]);

} catch (NumberFormatException e) {

System.out.println("Error: All arguments must be integers.");

return;

}

}

// Check if all three numbers are equal

if (numbers[0] == numbers[1] && numbers[0] == numbers[2]) {

System.out.println("equal");

} else {

System.out.println("not equal");

}

}

}

**2.Bank Account Management System**

1. **Description:** Write a program to manage bank accounts. Each account has an account number, a balance, and an account type (Savings or Checking). The program should allow depositing, withdrawing, and viewing the account balance. Savings accounts should earn interest monthly at a rate of 4%, while checking accounts have a minimum balance requirement of $500, and if the balance falls below this, a fee of $50 is charged.
2. **Test Cases:**
   * Deposit $500 into a savings account with a balance of $1000.
   * Withdraw $200 from a checking account with a balance of $600.
   * View the balance of a savings account after applying monthly interest.

Withdraw $100 from a checking account with a balance of $400 (check for minimum balance and fee)

PROGRAMM:-

public abstract class BankAccount {

protected int accountNumber;

protected double balance;

protected String accountType;

public BankAccount(int accountNumber, double balance, String accountType) {

this.accountNumber = accountNumber;

this.balance = balance;

this.accountType = accountType;

}

public void deposit(double amount) {

balance += amount;

System.out.println("Deposited $" + amount + ". New balance: $" + balance);

}

public abstract void withdraw(double amount);

public abstract void applyMonthlyInterestOrFee();

public void viewBalance() {

System.out.println("Account Number: " + accountNumber);

System.out.println("Account Type: " + accountType);

System.out.println("Balance: $" + balance);

}

}

// SavingsAccount.java

public class SavingsAccount extends BankAccount {

private static final double INTEREST\_RATE = 0.04;

public SavingsAccount(int accountNumber, double balance) {

super(accountNumber, balance, "Savings");

}

@Override

public void withdraw(double amount) {

if (amount > balance) {

System.out.println("Insufficient funds.");

} else {

balance -= amount;

System.out.println("Withdrew $" + amount + ". New balance: $" + balance);

}

}

@Override

public void applyMonthlyInterestOrFee() {

double interest = balance \* INTEREST\_RATE;

balance += interest;

System.out.println("Applied monthly interest of $" + interest + ". New balance: $" + balance);

}

}

// CheckingAccount.java

public class CheckingAccount extends BankAccount {

private static final double MINIMUM\_BALANCE = 500;

private static final double FEE = 50;

public CheckingAccount(int accountNumber, double balance) {

super(accountNumber, balance, "Checking");

}

@Override

public void withdraw(double amount) {

if (amount > balance) {

System.out.println("Insufficient funds.");

} else {

balance -= amount;

if (balance < MINIMUM\_BALANCE) {

balance -= FEE;

System.out.println("Withdrew $" + amount + ". New balance: $" + balance);

System.out.println("Minimum balance not met. Fee of $" + FEE + " applied.");

} else {

System.out.println("Withdrew $" + amount + ". New balance: $" + balance);

}

}

}

@Override

public void applyMonthlyInterestOrFee() {

// No interest or fee applied for checking accounts

}

}

// Main.java

public class Main {

public static void main(String[] args) {

SavingsAccount savingsAccount = new SavingsAccount(12345, 1000);

savingsAccount.deposit(500);

savingsAccount.applyMonthlyInterestOrFee();

savingsAccount.viewBalance();

CheckingAccount checkingAccount = new CheckingAccount(67890, 600);

checkingAccount.withdraw(200);

checkingAccount.viewBalance();

CheckingAccount checkingAccount2 = new CheckingAccount(34567, 400);

checkingAccount2.withdraw(100);

checkingAccount2.viewBalance();

}

}

3.Write a Java program to count the number of prime and composite numbers from a list of integers entered by the user. The program should handle edge cases such as negative numbers, non-integer inputs, and zero, and it should also handle an empty list gracefully.

Sample Input & Output:

Enter numbers (separated by spaces): 2 3 4 5 6 7 8 9 10 Number of prime numbers: 4

Number of composite numbers: 5 Test Cases:

Input: 11 13 17 19

Output:

Number of prime numbers: 4 Number of composite numbers: 0 Input: 4 6 8 9 10

Output:

Number of prime numbers: 0 Number of composite numbers: 5 Input: 0 1 2 3 5

Output:

Number of prime numbers: 3 Number of composite numbers: 0 Input: -7 -11 -13

Output:

Number of prime numbers: 0 Number of composite numbers: 0

PROGRAMM:-

import java.util.InputMismatchException;

import java.util.Scanner;

public class PrimeAndCompositeCounter {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int primeCount = 0;

int compositeCount = 0;

System.out.println("Enter a list of integers (space-separated):");

while (scanner.hasNext()) {

try {

int number = scanner.nextInt();

if (number < 0) {

System.out.println("Ignoring negative number: " + number);

} else if (number == 0 || number == 1) {

System.out.println("Ignoring neither prime nor composite number: " + number);

} else if (isPrime(number)) {

primeCount++;

} else {

compositeCount++;

}

} catch (InputMismatchException e) {

System.out.println("Ignoring non-integer input: " + scanner.next());

}

}

System.out.println("Prime numbers: " + primeCount);

System.out.println("Composite numbers: " + compositeCount);

}

private static boolean isPrime(int number) {

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

if (number % i == 0) {

return false;

}

}

return true;

}

}

4.Write a Java program to find the Mth maximum number and Nth minimum number in an array. After identifying these numbers, calculate the sum and difference between them. Ensure the program handles edge cases, such as invalid M or N values, appropriately.

Sample Input & Output:

Enter the array elements: [3, 1, 4, 9, 2, 7, 6] Enter the value of M (for Mth maximum): 2 Enter the value of N (for Nth minimum): 3 Mth maximum number: 7

Nth minimum number: 3 Sum: 10

Difference: 4

Test Cases:

Input:

Array: [5, 2, 8, 1, 9, 7, 3]

M = 1, N = 1

Output:

Mth maximum number: 9 Nth minimum number: 1 Sum: 10

Difference: 8 Input:

Array: [4, 8, 15, 16, 23, 42]

M = 3, N = 2

Output:

Mth maximum number: 16 Nth minimum number: 8 Sum: 24

Difference: 8 Input:

Array: [12, 7, 22, 14, 9]

M = 5, N = 1

Output:

Mth maximum number: 7 Nth minimum number: 7 Sum: 14

Difference: 0 Input:

Array: [6, 1, 3, 7, 2]

M = 0, N = 2

Output:

Invalid input for M or N

PROGRAMM:-

import java.util.Arrays;

import java.util.Scanner;

public class MaxMinCalculator {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Enter the array elements (space-separated):");

String[] arrayElements = scanner.nextLine().split(" ");

int[] array = new int[arrayElements.length];

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

array[i] = Integer.parseInt(arrayElements[i]);

}

System.out.println("Enter the value of M:");

int m = scanner.nextInt();

System.out.println("Enter the value of N:");

int n = scanner.nextInt();

if (m < 1 || n < 1) {

System.out.println("Invalid M or N value. M and N must be positive integers.");

return;

}

if (m > array.length || n > array.length) {

System.out.println("M or N is larger than the array size.");

return;

}

Arrays.sort(array);

int mthMax = array[array.length - m];

int nthMin = array[n - 1];

int sum = mthMax + nthMin;

int difference = mthMax - nthMin;

System.out.println("Mth maximum number: " + mthMax);

System.out.println("Nth minimum number: " + nthMin);

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

System.out.println("Difference: " + difference);

}

}

5.**Palindrome Number Check**

* **Description:** Write a Java program to check if a given integer is a palindrome. An integer is a palindrome if it reads the same backward as forward.
* **Input:** A single integer.
* **Output:** Print "Palindrome" if the number is a palindrome, otherwise print "Not a Palindrome".
* **Example:**
  + Input: 121
  + Output: Palindrome
  + Input: 123
  + Output: Not a Palindrome

PROGRAMM:-

import java.util.Scanner;

public class PalindromeChecker {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Enter an integer:");

int number = scanner.nextInt();

if (isPalindrome(number)) {

System.out.println("Palindrome");

} else {

System.out.println("Not a Palindrome");

}

}

private static boolean isPalindrome(int number) {

int reversedNumber = 0;

int originalNumber = number;

while (number != 0) {

int digit = number % 10;

reversedNumber = reversedNumber \* 10 + digit;

number /= 10;

}

return originalNumber == reversedNumber;

}

}

6.**ANANGRAM CHECKER:-**

**Description:** Write a Java program to check if two given strings are anagrams of each other. An anagram of a string is another string that contains the same characters, only the order of characters can be different.

* **Input:** Two strings.
* **Output:** Print "Anagrams" if the strings are anagrams, otherwise print "Not Anagrams".
* **Example:**
  + Input: "listen", "silent"
  + Output: Anagrams
  + Input: "hello", "world"
  + Output: Not Anagrams

PROGRAMM:-

import java.util.Arrays;

import java.util.Scanner;

public class AnagramChecker {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Enter the first string:");

String str1 = scanner.next();

System.out.println("Enter the second string:");

String str2 = scanner.next();

if (isAnagram(str1, str2)) {

System.out.println("Anagrams");

} else {

System.out.println("Not Anagrams");

}

}

private static boolean isAnagram(String str1, String str2) {

if (str1.length()!= str2.length()) {

return false;

}

char[] charArray1 = str1.toLowerCase().toCharArray();

char[] charArray2 = str2.toLowerCase().toCharArray();

Arrays.sort(charArray1);

Arrays.sort(charArray2);

return Arrays.equals(charArray1, charArray2);

}

}

7.**Find the Missing Number in an Array**

* **Description:** Given an array containing n-1 distinct numbers taken from the range 1 to n, find the missing number.
* **Input:** An array of integers.
* **Output:** The missing integer.
* **Example:**

o Input: [1, 2, 4, 6, 3, 7, 8]

o Output: 5

PROGRAMM:-

import java.util.Scanner;

public class MissingNumberFinder {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Enter the size of the array:");

int n = scanner.nextInt();

System.out.println("Enter the elements of the array:");

int[] array = new int[n - 1];

for (int i = 0; i < n - 1; i++) {

array[i] = scanner.nextInt();

}

int missingNumber = findMissingNumber(array, n);

System.out.println("The missing number is: " + missingNumber);

}

private static int findMissingNumber(int[] array, int n) {

int totalSum = n \* (n + 1) / 2;

int arraySum = 0;

for (int num : array) {

arraySum += num;

}

return totalSum - arraySum;

}

}

**8.Matrix Multiplication**

* **Description:** Write a Java program to multiply two matrices. The program should read two matrices and print their product.
* **Input:** Two 2D arrays (matrices).
* **Output:** The product matrix.
* **Example:**

o Input: A = [[1, 2], [3, 4]], B = [[2, 0],

[1, 2]]

o Output: [[4, 4], [10, 8]]

PROGRAMM:-

import java.util.Scanner;

public class MatrixMultiplier {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Enter the number of rows in the first matrix:");

int rowsA = scanner.nextInt();

System.out.println("Enter the number of columns in the first matrix:");

int colsA = scanner.nextInt();

System.out.println("Enter the elements of the first matrix:");

int[][] matrixA = new int[rowsA][colsA];

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

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

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

}

}

System.out.println("Enter the number of rows in the second matrix:");

int rowsB = scanner.nextInt();

System.out.println("Enter the number of columns in the second matrix:");

int colsB = scanner.nextInt();

System.out.println("Enter the elements of the second matrix:");

int[][] matrixB = new int[rowsB][colsB];

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

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

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

}

}

if (colsA != rowsB) {

System.out.println("Matrices cannot be multiplied.");

return;

}

int[][] productMatrix = multiplyMatrices(matrixA, matrixB);

System.out.println("The product matrix is:");

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

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

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

}

System.out.println();

}

}

private static int[][] multiplyMatrices(int[][] matrixA, int[][] matrixB) {

int rowsA = matrixA.length;

int colsA = matrixA[0].length;

int colsB = matrixB[0].length;

int[][] productMatrix = new int[rowsA][colsB];

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

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

for (int k = 0; k < colsA; k++) {

productMatrix[i][j] += matrixA[i][k] \* matrixB[k][j];

}

}

}

return productMatrix;

}

}

**9.Calculate the Frequency of Characters in a String**

* **Description:** Write a Java program to calculate the frequency of each character in a given string.
* **Input:** A single string.
* **Output:** Print each character and its frequency.
* **Example:**

o Input: "hello"

Output h: 1

e: 1

l: 2

o: 1

PROGRAMM:-

import java.util.HashMap;

import java.util.Map;

import java.util.Scanner;

public class CharacterFrequencyCalculator {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Enter a string:");

String inputString = scanner.next();

Map<Character, Integer> frequencyMap = calculateCharacterFrequency(inputString);

System.out.println("Character frequencies:");

for (Map.Entry<Character, Integer> entry : frequencyMap.entrySet()) {

System.out.println(entry.getKey() + ": " + entry.getValue());

}

}

private static Map<Character, Integer> calculateCharacterFrequency(String inputString) {

Map<Character, Integer> frequencyMap = new HashMap<>();

for (char c : inputString.toCharArray()) {

if (frequencyMap.containsKey(c)) {

frequencyMap.put(c, frequencyMap.get(c) + 1);

} else {

frequencyMap.put(c, 1);

}

}

return frequencyMap;

}

}

**10.Binary Search Implementation**

* **Description:** Implement the binary search algorithm to find the index of a target value within a sorted array.
* **Input:** A sorted array and a target value.
* **Output:** The index of the target value if found, otherwise -1.
* **Example:**

o Input: [1, 3, 5, 7, 9], 5

* + Output: 2

o Input: [1, 3, 5, 7, 9], 6

Output: -1

PROGRAMM:-

import java.util.Scanner;

public class BinarySearch {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Enter the size of the array:");

int size = scanner.nextInt();

System.out.println("Enter the elements of the array in sorted order:");

int[] array = new int[size];

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

array[i] = scanner.nextInt();

}

System.out.println("Enter the target value:");

int target = scanner.nextInt();

int index = binarySearch(array, target);

if (index != -1) {

System.out.println("Target value found at index " + index);

} else {

System.out.println("Target value not found in the array");

}

}

private static int binarySearch(int[] array, int target) {

int low = 0;

int high = array.length - 1;

while (low <= high) {

int mid = low + (high - low) / 2;

if (array[mid] == target) {

return mid;

} else if (array[mid] < target) {

low = mid + 1;

} else {

high = mid - 1;

}

}

return -1;

}

}

**11.Check for Balanced Parentheses**

* **Description:** Write a Java program to check whether the parentheses in a given expression are balanced.
* **Input:** A string containing parentheses.
* **Output:** Print "Balanced" if the parentheses are balanced, otherwise print "Not Balanced".
* **Example:**

o Input: "((()))"

* + Output: Balanced
  + Input: "(()"
  + Output: Not Balanced

PROGRAMM:-

import java.util.Stack;

import java.util.Scanner;

public class BalancedParentheses {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Enter an expression:");

String expression = scanner.next();

if (isBalanced(expression)) {

System.out.println("Balanced");

} else {

System.out.println("Not Balanced");

}

}

private static boolean isBalanced(String expression) {

Stack<Character> stack = new Stack<>();

for (char c : expression.toCharArray()) {

if (c == '(') {

stack.push(c);

} else if (c == ')') {

if (stack.isEmpty() || stack.pop() != '(') {

return false;

}

}

}

return stack.isEmpty();

}

}

**12.Remove Duplicates from a Sorted Array**

* **Description:** Write a Java program to remove duplicates from a sorted array and return the new length of the array.
* **Input:** A sorted array of integers.
* **Output:** The new length of the array after removing duplicates.
* **Example:**Input: [1, 1, 2, 2, 3]
* Output: 3 (the array becomes [1, 2, 3])

PROGRAMM:-

import java.util.Scanner;

public class RemoveDuplicates {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Enter the size of the array:");

int size = scanner.nextInt();

System.out.println("Enter the elements of the array in sorted order:");

int[] array = new int[size];

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

array[i] = scanner.nextInt();

}

int newLength = removeDuplicates(array);

System.out.println("New length of the array: " + newLength);

System.out.print("Array after removing duplicates: ");

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

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

}

}

private static int removeDuplicates(int[] array) {

if (array.length == 0) {

return 0;

}

int i = 0;

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

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

i++;

array[i] = array[j];

}

}

return i + 1;

}

}

**13.Calculate Factorial Recursively**

* **Description:** Write a recursive Java program to calculate the factorial of a given number.
* **Input:** A single integer.
* **Output:** The factorial of the integer.
* **Example:**
  + Input: 5
  + Output: 120
  + Input: 0
  + Output: 1

PROGRAMM:-

import java.util.Scanner;

public class FactorialRecursive {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

int number = scanner.nextInt();

long factorial = calculateFactorial(number);

System.out.println("Factorial of " + number + " is " + factorial);

}

private static long calculateFactorial(int number) {

if (number == 0 || number == 1) {

return 1;

} else {

return number \* calculateFactorial(number - 1);

}

}

}

**14.Longest Common Subsequence**

* **Description:** Write a Java program to find the length of the longest common subsequence between two strings.
* **Input:** Two strings.
* **Output:** The length of the longest common subsequence.
* **Example:**
  + Input: "abcde", "ace"

Output: 3 (LCS is "ace")

PROGRAMM:-

import java.util.Scanner;

public class LongestCommonSubsequence {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Enter the first string:");

String str1 = scanner.next();

System.out.println("Enter the second string:");

String str2 = scanner.next();

int lcsLength = findLCSLength(str1, str2);

System.out.println("Length of the longest common subsequence: " + lcsLength);

}

private static int findLCSLength(String str1, String str2) {

int m = str1.length();

int n = str2.length();

int[][] dp = new int[m + 1][n + 1];

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

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

if (i == 0 || j == 0) {

dp[i][j] = 0;

} else if (str1.charAt(i - 1) == str2.charAt(j - 1)) {

dp[i][j] = dp[i - 1][j - 1] + 1;

} else {

dp[i][j] = Math.max(dp[i - 1][j], dp[i][j - 1]);

}

}

}

return dp[m][n];

}

}

**15.Reverse a String Without Using Built-in Functions**

* **Description:** Write a Java program to reverse a given string without using any built-in string manipulation functions.
* **Input:** A single string.
* **Output:** The reversed string.
* **Example:**
* Input: "hello"
* Output: "olleh"

PROGRAMM:-

import java.util.Scanner;

public class ReverseString {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Enter a string:");

String str = scanner.next();

char[] charArray = str.toCharArray();

int left = 0;

int right = charArray.length - 1;

while (left < right) {

char temp = charArray[left];

charArray[left] = charArray[right];

charArray[right] = temp;

left++;

right--;

}

System.out.println("Reversed string: " + new String(charArray));

}

}

16.**Check Armstrong Number**

* **Description:** Write a Java program to check if a given number is an Armstrong number. An Armstrong number for a given number of digits is a number whose sum of its own digits each raised to the power of the number of digits is equal to the number itself.
* **Input:** A single integer
* **Output:** Print "Armstrong" if the number is an Armstrong number, otherwise print "Not Armstrong".
* **Example:**
  + Input: 153
  + Output: Armstrong

PROGRAMM:-

import java.util.Scanner;

public class ArmstrongNumber {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

int num = scanner.nextInt();

if (isArmstrong(num)) {

System.out.println("Armstrong");

} else {

System.out.println("Not Armstrong");

}

}

private static boolean isArmstrong(int num) {

int originalNum = num;

int sum = 0;

int digitCount = countDigits(num);

while (num != 0) {

int digit = num % 10;

sum += Math.pow(digit, digitCount);

num /= 10;

}

return sum == originalNum;

}

private static int countDigits(int num) {

int count = 0;

while (num != 0) {

num /= 10;

count++;

}

return count;

}

}

**17.Count Vowels and Consonants in a String**

* **Description:** Write a Java program to count the number of vowels and consonants in a given string.
* **Input:** A single string.
* **Output:** Print the count of vowels and consonants.
* **Example:**
  + Input: "OpenAI"
  + Output: Vowels: 3, Consonants: 3

PROGRAMM:-

import java.util.Scanner;

public class VowelsAndConsonants {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Enter a string:");

String str = scanner.next();

int vowelCount = 0;

int consonantCount = 0;

for (char c : str.toLowerCase().toCharArray()) {

if (c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u') {

vowelCount++;

} else if (c >= 'a' && c <= 'z') {

consonantCount++;

}

}

System.out.println("Vowels: " + vowelCount + ", Consonants: " + consonantCount);

}

}

**18.Find the GCD (Greatest Common Divisor) of Two Numbers**

* **Description:** Write a Java program to find the greatest common divisor (GCD) of two given numbers using Euclid's algorithm.
* **Input:** Two integers.
* **Output:** The GCD of the two integers.
* **Example:**
  + Input: 48, 18
  + Output: 6

PROGRAMM:-

import java.util.Scanner;

public class GCD {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Enter the first number:");

int num1 = scanner.nextInt();

System.out.println("Enter the second number:");

int num2 = scanner.nextInt();

int gcd = findGCD(num1, num2);

System.out.println("The GCD of " + num1 + " and " + num2 + " is " + gcd);

}

private static int findGCD(int num1, int num2) {

if (num2 == 0) {

return num1;

} else {

return findGCD(num2, num1 % num2);

}

}

}

**19.Bubble Sort Implementation**

* **Description:** Write a Java program to implement the bubble sort algorithm to sort an array of integers in ascending order.
* **Input:** An unsorted array of integers.
* **Output:** The sorted array.
* **Example:**

o Input: [64, 34, 25, 12, 22, 11, 90]

o Output: [11, 12, 22, 25, 34, 64, 90]

PROGRAMM:-

public class BubbleSort {

public static void main(String[] args) {

int[] arr = {64, 34, 25, 12, 22, 11, 90};

System.out.println("Original array:");

printArray(arr);

bubbleSort(arr);

System.out.println("Sorted array:");

printArray(arr);

}

private static void bubbleSort(int[] arr) {

int n = arr.length;

boolean swapped;

for (int i = 0; i < n - 1; i++) {

swapped = false;

for (int j = 0; j < n - i - 1; j++) {

if (arr[j] > arr[j + 1]) {

// Swap arr[j] and arr[j + 1]

int temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

swapped = true;

}

}

// If no two elements were swapped in the inner loop, the array is sorted

if (!swapped) {

break;

}

}

}

private static void printArray(int[] arr) {

for (int i : arr) {

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

}

System.out.println();

}

}

**20.Check if a String is a Valid Palindrome (Ignoring Spaces and Case)**

* **Description:** Write a Java program to check if a given string is a palindrome, ignoring spaces and case.
* **Input:** A single string.
* **Output:** Print "Palindrome" if the string is a palindrome, otherwise print "Not a Palindrome".
* **Example:**
  + Input: "A man a plan a canal Panama"
  + Output: Palindrome

PROGRAMM:-

public class PalindromeChecker {

public static void main(String[] args) {

String input = "A man a plan a canal Panama";

if (isPalindrome(input)) {

System.out.println("Palindrome");

} else {

System.out.println("Not a Palindrome");

}

}

private static boolean isPalindrome(String str) {

str = str.replaceAll("\\s+", "").toLowerCase(); // Remove spaces and convert to lowercase

int left = 0;

int right = str.length() - 1;

while (left < right) {

if (str.charAt(left) != str.charAt(right)) {

return false;

}

left++;

right--;

}

return true;

}

}

**21.Find All Prime Numbers Up to a Given Number**

* **Description:** Write a Java program to find all prime numbers up to a given number using the Sieve of Eratosthenes.
* **Input:** A single integer n.
* **Output:** A list of prime numbers less than or equal to n.
* **Example:**

o Input: 30

o Output: [2, 3, 5, 7, 11, 13, 17, 19, 23,29]

PROGRAMM:-

import java.util.ArrayList;

import java.util.List;

import java.util.Scanner;

public class SieveOfEratosthenes {

public static List<Integer> sieveOfEratosthenes(int n) {

boolean[] isPrime = new boolean[n + 1];

List<Integer> primes = new ArrayList<>();

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

isPrime[i] = true;

}

for (int p = 2; p \* p <= n; p++) {

if (isPrime[p]) {

for (int i = p \* p; i <= n; i += p) {

isPrime[i] = false;

}

}

}

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

if (isPrime[i]) {

primes.add(i);

}

}

return primes;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

int n = scanner.nextInt();

scanner.close();

List<Integer> primes = sieveOfEratosthenes(n);

System.out.println("Prime numbers less than or equal to " + n + ": " + primes);

}

}

**22.Remove Duplicates from a String**

* **Description:** Write a Java program to remove duplicate characters from a given string.
* **Input:** A single string.
* **Output:** The string with duplicates removed.
* **Example:**
  + Input: "programming"
  + Output: "progamin"

PROGRAMM:-

import java.util.LinkedHashSet;

import java.util.Scanner;

import java.util.Set;

public class RemoveDuplicates {

public static String removeDuplicates(String input) {

Set<Character> seenCharacters = new LinkedHashSet<>();

StringBuilder result = new StringBuilder();

for (char c : input.toCharArray()) {

if (!seenCharacters.contains(c)) {

seenCharacters.add(c);

result.append(c);

}

}

return result.toString();

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

String input = scanner.nextLine();

scanner.close();

String result = removeDuplicates(input);

System.out.println("String after removing duplicates: " + result);

}

}

**23.Find the Second Largest Element in an Array**

* **Description:** Write a Java program to find the second largest element in an array of integers.
* **Input:** An array of integers.
* **Output:** The second largest element.
* **Example:**

o Input: [12, 35, 1, 10, 34, 1]

* + o Output: 34

PROGRAMM:-

import java.util.Scanner;

public class SecondLargestElement {

public static int findSecondLargest(int[] array) {

if (array.length < 2) {

throw new IllegalArgumentException("Array must contain at least two elements");

}

int firstLargest = Integer.MIN\_VALUE;

int secondLargest = Integer.MIN\_VALUE;

for (int num : array) {

if (num > firstLargest) {

secondLargest = firstLargest;

firstLargest = num;

} else if (num > secondLargest && num != firstLargest) {

secondLargest = num;

}

}

if (secondLargest == Integer.MIN\_VALUE) {

throw new IllegalArgumentException("Array does not contain a second distinct largest

element");

}

return secondLargest;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

int n = scanner.nextInt();

int[] array = new int[n];

System.out.println("Enter the elements of the array:");

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

array[i] = scanner.nextInt();

}

scanner.close();

try {

int secondLargest = findSecondLargest(array);

System.out.println("The second largest element is: " + secondLargest);

} catch (IllegalArgumentException e) {

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

}

}

}

**24.Find the Longest Substring Without Repeating Characters**

* **Description:** Write a Java program to find the length of the longest substring without repeating characters in a given string.
* **Input:** A single string.
* **Output:** The length of the longest substring without repeating characters.
* **Example:**
  + Input: "abcabcbb"
  + Output: 3 (substring is "abc")

PROGRAMM:-

import java.util.HashSet;

import java.util.Set;

public class LongestSubstringWithoutRepeating {

public static void main(String[] args) {

// Example input

String input = "abcabcbb";

int result = lengthOfLongestSubstring(input);

System.out.println("Length of the longest substring without repeating characters: " + result);

}

public static int lengthOfLongestSubstring(String s) {

Set<Character> charSet = new HashSet<>();

int left = 0, right = 0;

int maxLength = 0;

while (right < s.length()) {

char rightChar = s.charAt(right);

// If the character is not in the set, add it

if (!charSet.contains(rightChar)) {

charSet.add(rightChar);

right++;

// Update the maximum length

maxLength = Math.max(maxLength, right - left);

} else {

// Remove the leftmost character and move the left pointer

char leftChar = s.charAt(left);

charSet.remove(leftChar);

left++;

}

}

return maxLength;

}

}

**25.Calculate Power Without Using Built-in Functions**

* **Description:** Write a Java program to calculate x raised to the power y (x^y) without using built-in math functions.
* **Input:** Two integers x and y.
* **Output:** The result of x^y.
* **Example:**
  + Input: 2, 3
  + Output: 8

PROGRAMM:-

import java.util.Scanner;

public class PowerCalculator {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Read input values for x and y

System.out.print("Enter the base (x): ");

int x = scanner.nextInt();

System.out.print("Enter the exponent (y): ");

int y = scanner.nextInt();

// Calculate x^y

long result = power(x, y);

// Output the result

System.out.println(x + " raised to the power " + y + " is: " + result);

}

public static long power(int x, int y) {

if (y == 0) {

return 1;

}

long result = 1;

boolean isNegative = false;

if (y < 0) {

isNegative = true;

y = -y;

}

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

result \*= x;

}

if (isNegative) {

return 1 / result;

}

return result;

}

}

**26.Merge Two Sorted Arrays**

* **Description:** Write a Java program to merge two sorted arrays into a single sorted array.
* **Input:** Two sorted arrays of integers.
* **Output:** A merged sorted array.
* **Example:**

o Input: [1, 3, 5], [2, 4, 6]

* + o Output: [1, 2, 3, 4, 5, 6]

PROGRAMM:-

import java.util.Arrays;

public class MergeSortedArrays {

public static void main(String[] args) {

// Example input

int[] arr1 = {1, 3, 5, 7};

int[] arr2 = {2, 4, 6, 8};

// Merging the two sorted arrays

int[] mergedArray = mergeArrays(arr1, arr2);

// Output the result

System.out.println("Merged sorted array: " + Arrays.toString(mergedArray));

}

public static int[] mergeArrays(int[] arr1, int[] arr2) {

int n1 = arr1.length;

int n2 = arr2.length;

int[] mergedArray = new int[n1 + n2];

int i = 0, j = 0, k = 0;

// Merge the arrays

while (i < n1 && j < n2) {

if (arr1[i] <= arr2[j]) {

mergedArray[k] = arr1[i];

i++;

} else {

mergedArray[k] = arr2[j];

j++;

}

k++;

}

// Copy remaining elements of arr1, if any

while (i < n1) {

mergedArray[k] = arr1[i];

i++;

k++;

}

// Copy remaining elements of arr2, if any

while (j < n2) {

mergedArray[k] = arr2[j];

j++;

k++;

}

return mergedArray;

}

}

**27.Check if Two Strings are Rotation of Each Other**

* **Description:** Write a Java program to check if two strings are rotations of each other.
* **Input:** Two strings.
* **Output:** Print "Rotation" if one string is a rotation of the other, otherwise print "Not Rotation".
* **Example:**
  + Input: "ABCD", "CDAB"
  + Output: Rotation

PROGRAMM:-

import java.util.Scanner;

public class StringRotationChecker {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Read input strings

System.out.print("Enter the first string: ");

String str1 = scanner.nextLine();

System.out.print("Enter the second string: ");

String str2 = scanner.nextLine();

// Check if the second string is a rotation of the first

if (areRotations(str1, str2)) {

System.out.println("Rotation");

} else {

System.out.println("Not Rotation");

}

}

public static boolean areRotations(String str1, String str2) {

// Check if lengths of the strings are equal and str2 is a substring of str1 concatenated with itself

return str1.length() == str2.length() && (str1 + str1).contains(str2);

}

}

**28.Fibonacci Series Using Recursion**

* **Description:** Write a Java program to print the first n

Fibonacci numbers using recursion.

* **Input:** An integer n.
* **Output:** The first n Fibonacci numbers.
* **Example:**

o Input: 7

* + Output: [0, 1, 1, 2, 3, 5, 8]

PROGRAMM:-

import java.util.Scanner;

public class FibonacciRecursion {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Read input value for n

System.out.print("Enter the number of Fibonacci numbers to print: ");

int n = scanner.nextInt();

// Print the first n Fibonacci numbers

System.out.println("The first " + n + " Fibonacci numbers are:");

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

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

}

}

public static int fibonacci(int n) {

if (n <= 1) {

return n;

}

return fibonacci(n - 1) + fibonacci(n - 2);

}

}

**29.Check if a Number is Perfect**

* **Description:** Write a Java program to check if a given number is a perfect number. A perfect number is a positive integer that is equal to the sum of its proper divisors, excluding itself.
* **Input:** A single integer.
* **Output:** Print "Perfect Number" if the number is perfect, otherwise print "Not a Perfect Number".
* **Example:**
  + Input: 28
  + Output: Perfect Number

PROGRAMM:-

import java.util.Scanner;

public class PerfectNumberChecker {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Read input value

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

int number = scanner.nextInt();

// Check if the number is a perfect number

if (isPerfectNumber(number)) {

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

} else {

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

}

}

public static boolean isPerfectNumber(int number) {

if (number <= 1) {

return false; // Perfect numbers are greater than 1

}

int sum = 0;

// Find all proper divisors and sum them

for (int i = 1; i <= number / 2; i++) {

if (number % i == 0) {

sum += i;

}

}

// Check if the sum of proper divisors equals the number

return sum == number;

}

}

**30.** Java program that converts a given decimal number into its binary and octal equivalents. The program also includes input validation to ensure the user enters a valid non-negative integer.

**Test Cases:**

1. **Test Case 1:**
   * **Input:** Decimal Number: 15
   * **Output:**
     + Binary Number = 1111
     + Octal Number = 17
2. **Test Case 2:**
   * **Input:** Decimal Number: 0
   * **Output:**
     + Binary Number = 0
     + Octal Number = 0
3. **Test Case 3:**
   * **Input:** Decimal Number: 255
   * **Output:**
     + Binary Number = 11111111
     + Octal Number = 377
4. **Test Case 4:**
   * **Input:** Decimal Number: -10
   * **Output:** Please enter a non-negative integer.
5. **Test Case 5:**
   * **Input:** Decimal Number: abc
   * **Output:** Invalid input. Please enter a valid integer.

PROGRAMM:-

import java.util.Scanner;

public class DecimalConverter {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Read input value

System.out.print("Enter a non-negative decimal number: ");

while (!scanner.hasNextInt()) {

System.out.println("Invalid input. Please enter a non-negative integer.");

scanner.next(); // Consume the invalid input

System.out.print("Enter a non-negative decimal number: ");

}

int decimalNumber = scanner.nextInt();

// Validate the input

if (decimalNumber < 0) {

System.out.println("Invalid input. Please enter a non-negative integer.");

} else {

// Convert to binary and octal

String binaryNumber = Integer.toBinaryString(decimalNumber);

String octalNumber = Integer.toOctalString(decimalNumber);

// Output the results

System.out.println("Binary Number = " + binaryNumber);

System.out.println("Octal Number = " + octalNumber);

}

}

}

**31.Student Grade Calculation**

* **Description:** Write a program that takes the marks of five subjects and calculates the total, average, and grade of a student. The grade is determined as follows: A for average >= 90, B for average >= 80 and < 90, C for average >= 70 and < 80, D for average >= 60 and < 70, and F for average < 60.
* **Test Cases:**
  1. Marks: 95, 92, 88, 90, 91 (Expected Grade: A)
  2. Marks: 75, 80, 72, 68, 74 (Expected Grade: C)
  3. Marks: 60, 59, 62, 58, 57 (Expected Grade: F)
  + Marks: 85, 89, 90, 87, 84 (Expected Grade: B)

PROGRAMM:-

import java.util.Scanner;

public class StudentGradeCalculator {

public static void main(String[] args) {

// Create a Scanner object for input

Scanner scanner = new Scanner(System.in);

// Array to store the marks of the five subjects

int[] marks = new int[5];

// Input marks for five subjects

System.out.println("Enter the marks for five subjects:");

for (int i = 0; i < 5; i++) {

System.out.print("Subject " + (i + 1) + ": ");

marks[i] = scanner.nextInt();

}

// Calculate total and average

int total = 0;

for (int mark : marks) {

total += mark;

}

double average = (double) total / marks.length;

// Determine the grade based on the average

char grade;

if (average >= 90) {

grade = 'A';

} else if (average >= 80) {

grade = 'B';

} else if (average >= 70) {

grade = 'C';

} else if (average >= 60) {

grade = 'D';

} else {

grade = 'F';

}

// Display the results

System.out.println("Total Marks: " + total);

System.out.println("Average Marks: " + average);

System.out.println("Grade: " + grade);

// Close the scanner

scanner.close();

}

}

**32.Electricity Bill Calculator**

* **Description:** Write a program to calculate the electricity bill based on the units consumed. The charges per unit are as follows:
  + First 100 units: $1.50 per unit
  + Next 200 units: $2.00 per unit
  + Above 300 units: $3.00 per unit
* Additionally, if the total bill exceeds $500, a surcharge of 10% is added.
* **Test Cases:**

1. Units Consumed: 150 (Expected Bill: $275)
2. Units Consumed: 350 (Expected Bill: $675 + 10% surcharge = $742.50)
3. Units Consumed: 90 (Expected Bill: $135)
   * Units Consumed: 600 (Expected Bill: $1500 + 10% surcharge = $1650)

PROGRAMM:-

import java.util.Scanner;

public class ElectricityBillCalculator {

public static void main(String[] args) {

// Create a Scanner object for input

Scanner scanner = new Scanner(System.in);

// Input units consumed

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

int units = scanner.nextInt();

// Calculate the bill based on the units consumed

double bill = 0.0;

if (units <= 100) {

bill = units \* 1.50;

} else if (units <= 300) {

bill = (100 \* 1.50) + ((units - 100) \* 2.00);

} else {

bill = (100 \* 1.50) + (200 \* 2.00) + ((units - 300) \* 3.00);

}

// Apply surcharge if bill exceeds $500

if (bill > 500) {

bill += bill \* 0.10; // Adding 10% surcharge

}

// Display the result

System.out.printf("Total Bill: $%.2f%n", bill);

// Close the scanner

scanner.close();

}

}

**33.Library Management System**

* **Description:** Write a program to manage a library system. The system should track books, which have a title, author, and availability status. It should allow users to borrow and return books, and it should ensure that books are not borrowed if they are already checked out.
* **Test Cases:**
  1. Borrow a book that is available.
  2. Try to borrow a book that is already borrowed.
  3. Return a book that was borrowed.
  + List all available books in the library

PROGRAMM:-

import java.util.ArrayList;

import java.util.List;

import java.util.Scanner;

class Book {

private String title;

private String author;

private boolean isAvailable;

public Book(String title, String author) {

this.title = title;

this.author = author;

this.isAvailable = true; // Books are available by default when added

}

public String getTitle() {

return title;

}

public String getAuthor() {

return author;

}

public boolean isAvailable() {

return isAvailable;

}

public void borrow() {

if (isAvailable) {

isAvailable = false;

System.out.println("You have borrowed: " + title);

} else {

System.out.println("Sorry, the book \"" + title + "\" is already borrowed.");

}

}

public void returnBook() {

if (!isAvailable) {

isAvailable = true;

System.out.println("You have returned: " + title);

} else {

System.out.println("The book \"" + title + "\" was not borrowed.");

}

}

@Override

public String toString() {

return title + " by " + author + " (Available: " + isAvailable + ")";

}

}

class Library {

private List<Book> books;

public Library() {

books = new ArrayList<>();

}

public void addBook(Book book) {

books.add(book);

System.out.println("Book added: " + book);

}

public void borrowBook(String title) {

for (Book book : books) {

if (book.getTitle().equalsIgnoreCase(title)) {

book.borrow();

return;

}

}

System.out.println("Book with title \"" + title + "\" not found.");

}

public void returnBook(String title) {

for (Book book : books) {

if (book.getTitle().equalsIgnoreCase(title)) {

book.returnBook();

return;

}

}

System.out.println("Book with title \"" + title + "\" not found.");

}

public void listAvailableBooks() {

System.out.println("Available Books:");

for (Book book : books) {

if (book.isAvailable()) {

System.out.println(book);

}

}

}

}

public class LibrarySystem {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

Library library = new Library();

// Adding some books to the library

library.addBook(new Book("1984", "George Orwell"));

library.addBook(new Book("To Kill a Mockingbird", "Harper Lee"));

library.addBook(new Book("The Great Gatsby", "F. Scott Fitzgerald"));

while (true) {

System.out.println("\nLibrary System");

System.out.println("1. Borrow a book");

System.out.println("2. Return a book");

System.out.println("3. List available books");

System.out.println("4. Exit");

System.out.print("Choose an option: ");

int choice = scanner.nextInt();

scanner.nextLine(); // Consume newline

switch (choice) {

case 1:

System.out.print("Enter the title of the book to borrow: ");

String borrowTitle = scanner.nextLine();

library.borrowBook(borrowTitle);

break;

case 2:

System.out.print("Enter the title of the book to return: ");

String returnTitle = scanner.nextLine();

library.returnBook(returnTitle);

break;

case 3:

library.listAvailableBooks();

break;

case 4:

System.out.println("Exiting the library system.");

scanner.close();

return;

default:

System.out.println("Invalid choice. Please try again.");

break;

}

}

}

}

**34.Tax Calculator**

* **Description:** Write a program to calculate income tax based on the following slab:
  + Income up to $10,000: No tax

o Income from $10,001 to $20,000: 10% tax

o Income from $20,001 to $50,000: 20% tax

* + Income above $50,000: 30% tax
* The program should take the user’s income as input and calculate the tax accordingly.
* **Test Cases:**

1. Income: $9,000 (Expected Tax: $0)
2. Income: $15,000 (Expected Tax: $500)
3. Income: $35,000 (Expected Tax: $4,000)

4. Income: $70,000 (Expected Tax: $15,000)

PROGRAMM:-

import java.util.Scanner;

public class IncomeTaxCalculator {

public static void main(String[] args) {

// Create a Scanner object for input

Scanner scanner = new Scanner(System.in);

// Input the user's income

System.out.print("Enter your income: $");

double income = scanner.nextDouble();

// Calculate the tax based on the income slab

double tax = calculateTax(income);

// Display the result

System.out.printf("Your income tax is: $%.2f%n", tax);

// Close the scanner

scanner.close();

}

// Method to calculate tax based on income slabs

private static double calculateTax(double income) {

double tax = 0.0;

if (income <= 10000) {

// No tax for income up to $10,000

tax = 0.0;

} else if (income <= 20000) {

// 10% tax for income from $10,001 to $20,000

tax = (income - 10000) \* 0.10;

} else if (income <= 50000) {

// 20% tax for income from $20,001 to $50,000

tax = (10000 \* 0.10) + (income - 20000) \* 0.20;

} else {

// 30% tax for income above $50,000

tax = (10000 \* 0.10) + (30000 \* 0.20) + (income - 50000) \* 0.30;

}

return tax;

}

}

**35.Simple ATM System**

* **Description:** Create a simple ATM system where users can check their balance, deposit money, and withdraw money. The system should ensure that withdrawals do not exceed the available balance and should update the balance after each transaction.
* **Test Cases:**
  1. Initial Balance: $5,000, Withdraw $1,000 (Expected Balance: $4,000)
  2. Deposit $2,000 (Expected Balance: $6,000)
  3. Try to withdraw $10,000 (Expected Output: Insufficient funds)
  + Check balance after transactions

PROGRAMM:-

import java.util.Scanner;

public class ATM {

private double balance;

public ATM() {

this.balance = 0.0;

}

public double getBalance() {

return balance;

}

public void deposit(double amount) {

if (amount > 0) {

balance += amount;

System.out.printf("Deposited: $%.2f%n", amount);

} else {

System.out.println("Deposit amount must be positive.");

}

}

public void withdraw(double amount) {

if (amount > 0) {

if (amount <= balance) {

balance -= amount;

System.out.printf("Withdrew: $%.2f%n", amount);

} else {

System.out.println("Insufficient funds.");

}

} else {

System.out.println("Withdrawal amount must be positive.");

}

}

public void displayMenu() {

System.out.println("\nATM Menu:");

System.out.println("1. Check Balance");

System.out.println("2. Deposit");

System.out.println("3. Withdraw");

System.out.println("4. Exit");

}

public void run() {

Scanner scanner = new Scanner(System.in);

while (true) {

displayMenu();

System.out.print("Enter choice (1-4): ");

int choice = scanner.nextInt();

switch (choice) {

case 1:

System.out.printf("Current Balance: $%.2f%n", getBalance());

break;

case 2:

System.out.print("Enter deposit amount: $");

double depositAmount = scanner.nextDouble();

deposit(depositAmount);

break;

case 3:

System.out.print("Enter withdrawal amount: $");

double withdrawAmount = scanner.nextDouble();

withdraw(withdrawAmount);

break;

case 4:

System.out.println("Exiting...");

scanner.close();

return;

default:

System.out.println("Invalid choice. Please select a valid option.");

}

}

}

public static void main(String[] args) {

ATM atm = new ATM();

atm.run();

}

}

**36.Payroll System**

* **Description:** Develop a payroll system that calculates the net salary of employees. The salary should include base pay and various allowances (house rent allowance, dearness allowance) and deductions (tax, provident fund). The allowances are a percentage of the base pay, and the deductions are either fixed or percentage-based.
* **Test Cases:**

1. Base Pay: $5,000, HRA: 20%, DA: 10%,

Tax:10%, PF: $500 (Expected Net Salary: $5,900)

2. Base Pay: $10,000, HRA: 25%, DA: 15%,

Tax:12%, PF: $1000 (Expected Net Salary: $11,200)

3. Calculate net salary for different combinations of allowances and deductions.

PROGRAMM:-

import java.util.Scanner;

public class PayrollSystem {

private double basePay;

private double houseRentAllowancePercentage;

private double dearnessAllowancePercentage;

private double taxPercentage;

private double providentFundPercentage;

public PayrollSystem(double basePay, double houseRentAllowancePercentage, double dearnessAllowancePercentage, double taxPercentage, double providentFundPercentage) {

this.basePay = basePay;

this.houseRentAllowancePercentage = houseRentAllowancePercentage;

this.dearnessAllowancePercentage = dearnessAllowancePercentage;

this.taxPercentage = taxPercentage;

this.providentFundPercentage = providentFundPercentage;

}

public double calculateHouseRentAllowance() {

return basePay \* houseRentAllowancePercentage / 100;

}

public double calculateDearnessAllowance() {

return basePay \* dearnessAllowancePercentage / 100;

}

public double calculateTax() {

return basePay \* taxPercentage / 100;

}

public double calculateProvidentFund() {

return basePay \* providentFundPercentage / 100;

}

public double calculateNetSalary() {

double allowances = calculateHouseRentAllowance() + calculateDearnessAllowance();

double deductions = calculateTax() + calculateProvidentFund();

return basePay + allowances - deductions;

}

public void displaySalaryBreakdown() {

System.out.printf("Base Pay: $%.2f%n", basePay);

System.out.printf("House Rent Allowance (%.2f%%): $%.2f%n", houseRentAllowancePercentage, calculateHouseRentAllowance());

System.out.printf("Dearness Allowance (%.2f%%): $%.2f%n", dearnessAllowancePercentage, calculateDearnessAllowance());

System.out.printf("Tax (%.2f%%): $%.2f%n", taxPercentage, calculateTax());

System.out.printf("Provident Fund (%.2f%%): $%.2f%n", providentFundPercentage, calculateProvidentFund());

System.out.printf("Net Salary: $%.2f%n", calculateNetSalary());

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter base pay: $");

double basePay = scanner.nextDouble();

System.out.print("Enter house rent allowance percentage: ");

double houseRentAllowancePercentage = scanner.nextDouble();

System.out.print("Enter dearness allowance percentage: ");

double dearnessAllowancePercentage = scanner.nextDouble();

System.out.print("Enter tax percentage: ");

double taxPercentage = scanner.nextDouble();

System.out.print("Enter provident fund percentage: ");

double providentFundPercentage = scanner.nextDouble();

PayrollSystem payroll = new PayrollSystem(

basePay, houseRentAllowancePercentage, dearnessAllowancePercentage, taxPercentage, providentFundPercentage

);

payroll.displaySalaryBreakdown();

scanner.close();

}

}

**37.Online Shopping Cart**

* **Description:** Implement an online shopping cart where users can add and remove items, view the total price of items in the cart, and apply discounts if applicable (e.g., a discount code). The program should handle different product prices and quantities.
* **Test Cases:**
  1. Add items to the cart and check the total.
  2. Remove an item from the cart and update the total.
  3. Apply a discount code (e.g., 10% off) and check the updated total.
  + Empty the cart and check that the total is $0.

PROGRAMM:-

import java.util.HashMap;

import java.util.Map;

import java.util.Scanner;

public class ShoppingCart {

private final Map<String, Item> cart;

private double totalPrice;

public ShoppingCart() {

cart = new HashMap<>();

totalPrice = 0.0;

}

public void addItem(String name, double price, int quantity) {

if (cart.containsKey(name)) {

Item item = cart.get(name);

item.setQuantity(item.getQuantity() + quantity);

item.setTotalPrice(item.getTotalPrice() + price \* quantity);

} else {

cart.put(name, new Item(name, price, quantity));

}

updateTotalPrice();

}

public void removeItem(String name, int quantity) {

if (cart.containsKey(name)) {

Item item = cart.get(name);

if (item.getQuantity() <= quantity) {

cart.remove(name);

} else {

item.setQuantity(item.getQuantity() - quantity);

item.setTotalPrice(item.getTotalPrice() - item.getPrice() \* quantity);

}

updateTotalPrice();

}

}

public void applyDiscount(String discountCode) {

// Example discount codes

double discount = 0.0;

if ("DISCOUNT10".equals(discountCode)) {

discount = 0.10;

} else if ("DISCOUNT20".equals(discountCode)) {

discount = 0.20;

}

totalPrice -= totalPrice \* discount;

}

public double getTotalPrice() {

return totalPrice;

}

public void viewCart() {

System.out.println("Items in your cart:");

for (Item item : cart.values()) {

System.out.println(item);

}

System.out.println("Total Price: $" + String.format("%.2f", totalPrice));

}

private void updateTotalPrice() {

totalPrice = 0.0;

for (Item item : cart.values()) {

totalPrice += item.getTotalPrice();

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

ShoppingCart cart = new ShoppingCart();

while (true) {

System.out.println("Choose an option:");

System.out.println("1. Add item");

System.out.println("2. Remove item");

System.out.println("3. View cart");

System.out.println("4. Apply discount");

System.out.println("5. Exit");

int choice = scanner.nextInt();

scanner.nextLine(); // Consume newline

switch (choice) {

case 1:

System.out.println("Enter item name:");

String name = scanner.nextLine();

System.out.println("Enter item price:");

double price = scanner.nextDouble();

System.out.println("Enter quantity:");

int quantity = scanner.nextInt();

scanner.nextLine(); // Consume newline

cart.addItem(name, price, quantity);

break;

case 2:

System.out.println("Enter item name to remove:");

String itemName = scanner.nextLine();

System.out.println("Enter quantity to remove:");

int qtyToRemove = scanner.nextInt();

scanner.nextLine(); // Consume newline

cart.removeItem(itemName, qtyToRemove);

break;

case 3:

cart.viewCart();

break;

case 4:

System.out.println("Enter discount code:");

String discountCode = scanner.nextLine();

cart.applyDiscount(discountCode);

break;

case 5:

System.out.println("Exiting...");

scanner.close();

return;

default:

System.out.println("Invalid choice, please try again.");

}

}

}

}

class Item {

private final String name;

private final double price;

private int quantity;

private double totalPrice;

public Item(String name, double price, int quantity) {

this.name = name;

this.price = price;

this.quantity = quantity;

this.totalPrice = price \* quantity;

}

public String getName() {

return name;

}

public double getPrice() {

return price;

}

public int getQuantity() {

return quantity;

}

public void setQuantity(int quantity) {

this.quantity = quantity;

this.totalPrice = price \* quantity;

}

public double getTotalPrice() {

return totalPrice;

}

public void setTotalPrice(double totalPrice) {

this.totalPrice = totalPrice;

}

@Override

public String toString() {

return name + ": $" + String.format("%.2f", price) + " x " + quantity + " = $" + String.format("%.2f", totalPrice);

}

}

**38.**Write a Java program that safely handles the division of two numbers and specifically checks for division by zero. If division by zero is attempted, the program should catch the exception and provide an appropriate error message.

**Test Cases:**

1. **Test Case 1: Valid Division**
   * **Input:** Numerator: 10, Denominator: 2
   * **Expected Output:** Result: 5, Division operation complete.
2. **Test Case 2: Division by Zero**
   * **Input:** Numerator: 10, Denominator: 0
   * **Expected Output:** Error: Division by zero is not allowed., Division operation complete.
3. **Test Case 3: Negative Numbers**
   * **Input:** Numerator: -20, Denominator: 4
   * **Expected Output:** Result: -5, Division operation complete.
4. **Test Case 4: Zero Numerator**
   * **Input:** Numerator: 0, Denominator: 5
   * **Expected Output:** Result: 0, Division operation complete.
5. **Test Case 5: Large Numbers**
   * **Input:** Numerator: 1000000, Denominator: 1000
   * **Expected Output:** Result: 1000, Division operation complete

PROGRAMM:-

import java.util.InputMismatchException;

import java.util.Scanner;

public class SafeDivision {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

double numerator = 0;

double denominator = 0;

while (true) {

try {

// Input numerator

System.out.println("Enter the numerator:");

numerator = scanner.nextDouble();

// Input denominator

System.out.println("Enter the denominator:");

denominator = scanner.nextDouble();

// Perform division

double result = divide(numerator, denominator);

System.out.println("Result: " + result);

break; // Exit the loop after successful division

} catch (InputMismatchException e) {

// Handle invalid input

System.out.println("Invalid input. Please enter numeric values.");

scanner.next(); // Clear the invalid input

} catch (ArithmeticException e) {

// Handle division by zero

System.out.println("Error: Division by zero is not allowed.");

}

}

scanner.close();

}

public static double divide(double numerator, double denominator) {

if (denominator == 0) {

throw new ArithmeticException("Division by zero");

}

return numerator / denominator;

}

}

**39.Handling Multiple Exceptions**

**Scenario:**

Write a Java program that takes an array of integers from the user and an index position. The program should attempt to access the element at the given index and divide it by another number provided by the user. Handle the following exceptions:

* ArrayIndexOutOfBoundsException: If the index is out of the array bounds.
* ArithmeticException: If there is an attempt to divide by zero.
* Any other general exception.

**Test Case Scenarios:**

1. **Valid Input:**
   * **Input:** Array: [10, 20, 30], Index: 1, Divisor: 2
   * **Expected Output:** Result: 10, Execution complete.
2. **Array Index Out of Bounds:**
   * **Input:** Array: [10, 20, 30], Index: 5, Divisor: 2
   * **Expected Output:** Error: Index out of bounds, Execution complete.
3. **Division by Zero:**
   * **Input:** Array: [10, 20, 30], Index: 1, Divisor: 0
   * **Expected Output:** Error: Division by zero, Execution complete.
4. **Negative Index:**
   * **Input:** Array: [10, 20, 30], Index: -1, Divisor: 2
   * **Expected Output:** Error: Index out of bounds, Execution complete.
5. **Empty Array:**
   * **Input:** Array: [], Index: 0, Divisor: 1
   * **Expected Output:** Error: Index out of bounds, Execution complete.

PROGRAMM:-

import java.util.InputMismatchException;

import java.util.Scanner;

public class ArrayDivision {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Read array size

System.out.println("Enter the number of elements in the array:");

int size = readInt(scanner);

int[] array = new int[size];

// Read array elements

System.out.println("Enter " + size + " integers for the array:");

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

array[i] = readInt(scanner);

}

// Read index

System.out.println("Enter the index of the element to divide:");

int index = readInt(scanner);

// Read divisor

System.out.println("Enter the number to divide the element by:");

int divisor = readInt(scanner);

try {

// Access array element and perform division

int element = array[index];

double result = divide(element, divisor);

System.out.println("Result: " + result);

} catch (ArrayIndexOutOfBoundsException e) {

System.out.println("Error: Index " + e.getMessage() + " is out of bounds.");

} catch (ArithmeticException e) {

System.out.println("Error: Division by zero is not allowed.");

} catch (Exception e) {

System.out.println("An unexpected error occurred: " + e.getMessage());

} finally {

scanner.close();

}

}

private static int readInt(Scanner scanner) {

while (true) {

try {

return scanner.nextInt();

} catch (InputMismatchException e) {

System.out.println("Invalid input. Please enter an integer value.");

scanner.next(); // Clear the invalid input

}

}

}

private static double divide(int numerator, int divisor) {

if (divisor == 0) {

throw new ArithmeticException("Division by zero");

}

return (double) numerator / divisor;

}

}

**40.Custom Exception**

**Scenario:**

Write a Java program that simulates a bank account. The program should throw a custom InsufficientFundsException if a withdrawal amount exceeds the account balance. The program should also throw a custom NegativeAmountException if the withdrawal or deposit amount is negative.

**Test Case Scenarios:**

1. **Valid Deposit and Withdrawal:**
   * **Input:** Deposit: 500, Withdrawal: 200
   * **Expected Output:** Deposited: 500, Withdrew: 200, Current Balance: 1300
2. **Insufficient Funds:**
   * **Input:** Withdrawal: 1500
   * **Expected Output:** Exception: Insufficient funds for withdrawal.
3. **Negative Deposit:**
   * **Input:** Deposit: -50
   * **Expected Output:** Exception: Deposit amount cannot be negative.
4. **Negative Withdrawal:**
   * **Input:** Withdrawal: -100

PROGRAMM:-

import java.util.Scanner;

// Custom exception class

class InvalidAgeException extends Exception {

public InvalidAgeException(String message) {

super(message);

}

}

public class AgeValidation {

// Method to validate age

public static void validateAge(int age) throws InvalidAgeException {

if (age < 18) {

throw new InvalidAgeException("Age must be 18 or older. Provided age: " + age);

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

try {

// Input age

System.out.println("Enter your age:");

int age = scanner.nextInt();

// Validate age

validateAge(age);

// If no exception, print success message

System.out.println("Age validated successfully. You are allowed.");

} catch (InvalidAgeException e) {

// Handle the custom exception

System.out.println("Error: " + e.getMessage());

} catch (Exception e) {

// Handle any other exceptions

System.out.println("An unexpected error occurred: " + e.getMessage());

} finally {

scanner.close();

}

}

}

**42.Nested Try-Catch Blocks**

**Scenario:**

Write a Java program that demonstrates nested try-catch blocks, where the outer block handles a general exception and the inner block handles specific exceptions like ArithmeticException and ArrayIndexOutOfBoundsException.

**Test Cases:**

1. **No Exception:**

o **Array:** [10, 20, 2, 40]

* + **Output:** Result: 10, Execution continues...

1. **Arithmetic Exception:**

o **Array:** [10, 20, 0, 40]

* + **Output:** Inner catch: ArithmeticException caught, Execution continues...

1. **Array Index Out of Bounds:**

o **Array:** [10, 20, 30]

* + **Output:** Outer catch: ArrayIndexOutOfBoundsException caught, Execution continues…

PROGRAMM:-

public class BankAccount {

private double balance;

public BankAccount(double initialBalance) {

this.balance = initialBalance;

}

public void deposit(double amount) throws NegativeAmountException {

if (amount < 0) {

throw new NegativeAmountException("Deposit amount cannot be negative.");

}

balance += amount;

System.out.println("Deposited: " + amount);

}

public void withdraw(double amount) throws InsufficientFundsException, NegativeAmountException {

if (amount < 0) {

throw new NegativeAmountException("Withdrawal amount cannot be negative.");

}

if (amount > balance) {

throw new InsufficientFundsException("Insufficient funds for withdrawal.");

}

balance -= amount;

System.out.println("Withdrew: " + amount);

}

public double getBalance() {

return balance;

}

public static class InsufficientFundsException extends Exception {

public InsufficientFundsException(String message) {

super(message);

}

}

public static class NegativeAmountException extends Exception {

public NegativeAmountException(String message) {

super(message);

}

}

public static void main(String[] args) {

BankAccount account = new BankAccount(1000);

try {

account.deposit(500);

account.withdraw(200);

System.out.println("Current Balance: " + account.getBalance());

account.withdraw(1500);

account.deposit(-50);

account.withdraw(-100);

account.deposit(100);

account.withdraw(50);

account.withdraw(20);

System.out.println("Current Balance: " + account.getBalance());

} catch (InsufficientFundsException | NegativeAmountException e) {

System.out.println("Exception: " + e.getMessage());

}

}

}

**42.Try-With-Resources for Automatic Resource Management**

**Scenario:**

Write a Java program that reads from a file using try-with- resources, ensuring that the file resource is closed automatically even if an exception occurs.

**Test Cases:**

1. **Valid File Path:**
   * **File Content:** "Hello, World!"
   * **Output:** Hello, World!
2. **Invalid File Path:**

**File Path:** "nonexistent.txt"

o **Output:** Error reading the file: nonexistent.txt (No such file or directory)**Boundary**

PROGRAMM:-

public class NestedTryCatch {

public static void main(String[] args) {

int[] array = {10, 20, 2, 40}; // Test case 1: No Exception

// int[] array = {10, 20, 0, 40}; // Test case 2: Arithmetic Exception

// int[] array = {10, 20, 30}; // Test case 3: Array Index Out of Bounds

try {

try {

int result = array[0] / array[2]; // Potential ArithmeticException

System.out.println("Result: " + result);

} catch (ArithmeticException e) {

System.out.println("Inner catch: ArithmeticException caught");

} catch (ArrayIndexOutOfBoundsException e) {

System.out.println("Inner catch: ArrayIndexOutOfBoundsException caught");

}

} catch (Exception e) {

System.out.println("Outer catch: " + e.getClass().getSimpleName() + " caught");

}

System.out.println("Execution continues...");

}

}

**43.Try-With-Resources for Automatic Resource Management**

**Scenario:**

Write a Java program that reads from a file using try-with- resources, ensuring that the file resource is closed automatically even if an exception occurs.

**Test Cases:**

**1.Valid File Path:**

* + **File Content:** "Hello, World!"
  + **Output:** Hello, World!

**2.Invalid File Path:**

* + **File Path:** "nonexistent.txt"
  + **Expected Output:** Valid age for registration.
  + **Output:** Error reading the file: nonexistent.txt (No such file or directory)

PROGRAMM:-

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.IOException;

public class TryWithResources {

public static void main(String[] args) {

String filePath = "example.txt"; // Test case 1: Valid File Path

// String filePath = "nonexistent.txt"; // Test case 2: Invalid File Path

try (BufferedReader reader = new BufferedReader(new FileReader(filePath))) {

String line;

while ((line = reader.readLine()) != null) {

System.out.println(line);

}

} catch (IOException e) {

System.err.println("Error reading the file: " + filePath + " (" + e.getMessage() + ")");

}

}

}

**44.Custom Exception Chaining**

**Scenario:**

Write a Java program that demonstrates exception chaining, where a custom exception wraps another exception.

**Test Cases:**

1. **Exception Chaining Demonstration:**
   * **Input:** "invalid" string
   * **Output:** Caught: Invalid input encountered in method1, Original cause: NumberFormatException

PROGRAMM:-

public class CustomExceptionChaining {

public static void main(String[] args) {

try {

method1("invalid"); // Test case 1: Exception Chaining Demonstration

} catch (CustomException e) {

System.out.println("Caught: " + e.getMessage());

System.out.println("Original cause: " + e.getCause());

}

}

public static void method1(String input) throws CustomException {

try {

method2(input);

} catch (NumberFormatException e) {

throw new CustomException("Invalid input encountered in method1", e);

}

}

public static void method2(String input) throws NumberFormatException {

Integer.parseInt(input);

}

}

class CustomException extends Exception {

public CustomException(String message, Throwable cause) {

super(message, cause);

}

}

**45.Using Finally Block**

**Scenario:**

Write a Java program to demonstrate the use of a finally block, which executes regardless of whether an exception is thrown or not.

**Test Cases:**

1. **Exception Occurs:**
   * **Input:** Division by zero
   * **Output:** Exception caught, Finally block executed, Rest of the code.
2. **No Exception:**
   * **Input:** Valid division
   * **Output:** Result printed, Finally block executed, Rest of the code.

PROGRAMM:-

public class UsingFinallyBlock {

public static void main(String[] args) {

try {

int result = divide(10, 0); // Test case 1: Exception Occurs

// int result = divide(10, 2); // Test case 2: No Exception

System.out.println("Result: " + result);

} catch (ArithmeticException e) {

System.out.println("Exception caught");

} finally {

System.out.println("Finally block executed");

}

System.out.println("Rest of the code");

}

public static int divide(int a, int b) {

return a / b;

}

}

**46.Rethrowing an Exception**

**Scenario:**

Write a Java program that catches an exception, processes it, and then rethrows it to be handled by another catch block or by the calling method.

**Test Cases:**

1. **Rethrown Exception:**

* + **Input:** Division by zero
  + **Output:** Exception caught in method1, Exception rethrown and caught in main

PROGRAMM:-

public class RethrowingException {

public static void main(String[] args) {

try {

method1(10, 0); // Test case 1: Rethrown Exception

} catch (ArithmeticException e) {

System.out.println("Exception rethrown and caught in main.");

}

}

public static void method1(int a, int b) {

try {

int result = divide(a, b);

System.out.println("Result: " + result);

} catch (ArithmeticException e) {

System.out.println("Exception caught in method1.");

throw e; // Rethrow the exception

}

}

public static int divide(int a, int b) {

return a / b;

}

}

**47.Java Program to Print the First N Perfect Numbers**

**Scenario:**

Write a Java program that prints the first n perfect numbers. A perfect number is a positive integer that is equal to the sum of its proper divisors (excluding itself).

**Test Case Scenarios:**

1. **Valid Input - First 3 Perfect Numbers:**
   * **Input:** N = 3
   * **Output:** First 3 perfect numbers are: 6, 28, 496
2. **Valid Input - First 5 Perfect Numbers:**
   * **Input:** N = 5
   * **Output:** First 5 perfect numbers are: 6, 28, 496, 8128, 33550336
3. **Invalid Input - Zero or Negative Number:**
   * **Input:** N = 0
   * **Output:** The number must be a positive integer.
4. **Large Input - First 1 Perfect Number:**
   * **Input:** N = 1
   * **Output:** First 1 perfect number is: 6
5. **Edge Case - Input Greater Than 100:**
   * **Input:** N = 100
   * **Output:** (The program will print the first 100 perfect numbers if they exist within a reasonable range, otherwise it will print available perfect numbers within limits.)

PROGRAMM:-

import java.util.ArrayList;

import java.util.List;

public class PerfectNumbers {

public static void main(String[] args) {

int n = 3; // Test case 1: Valid Input - First 3 Perfect Numbers

// int n = 5; // Test case 2: Valid Input - First 5 Perfect Numbers

// int n = 0; // Test case 3: Invalid Input - Zero or Negative Number

// int n = 1; // Test case 4: Large Input - First 1 Perfect Number

// int n = 100; // Test case 5: Edge Case - Input Greater Than 100

if (n <= 0) {

System.out.println("The number must be a positive integer.");

return;

}

List<Integer> perfectNumbers = new ArrayList<>();

int num = 1;

while (perfectNumbers.size() < n) {

if (isPerfectNumber(num)) {

perfectNumbers.add(num);

}

num++;

}

System.out.print("First " + n + " perfect numbers are: ");

for (int i = 0; i < perfectNumbers.size(); i++) {

System.out.print(perfectNumbers.get(i));

if (i < perfectNumbers.size() - 1) {

System.out.print(", ");

}

}

}

public static boolean isPerfectNumber(int num) {

int sum = 0;

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

if (num % i == 0) {

sum += i;

}

}

return sum == num;

}

}

**48.Find the Largest and Smallest Element in an Array**

**Scenario:**

Write a Java program to find the largest and smallest elements in an array of integers.

**Test Case Scenarios:**

1. **Positive Numbers:**

o **Input:** Array: [3, 5, 7, 2, 8]

o **Output:** Largest element: 8, Smallest element: 2

1. **Negative Numbers:**
   * **Input:** Array: [-3, -5, -1, -8, -2]

o **Output:** Largest element: -1, Smallest element: - 8

3. **Mixed Numbers:**

o **Input:** Array: [3, -5, 7, 2, -8]

* + **Output:** Largest element: 7, Smallest element: - 8

PROGRAMM:-

public class LargestAndSmallestElements {

public static void main(String[] args) {

int[] array = {3, 5, 7, 2, 8}; // Test case 1: Positive Numbers

// int[] array = {-3, -5, -1, -8, -2}; // Test case 2: Negative Numbers

// int[] array = {3, -5, 7, 2, -8}; // Test case 3: Mixed Numbers

if (array.length == 0) {

System.out.println("Array is empty.");

return;

}

int largest = array[0];

int smallest = array[0];

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

if (array[i] > largest) {

largest = array[i];

}

if (array[i] < smallest) {

smallest = array[i];

}

}

System.out.println("Largest element: " + largest);

System.out.println("Smallest element: " + smallest);

}

}

**49.Reverse a String**

**Scenario:**

Write a Java program to reverse a given string.

**Test Case Scenarios:**

1. **Normal String:**
   * **Input:** "hello"
   * **Output:** "olleh"
2. **Empty String:**
   * **Input:** ""
   * **Output:** ""
3. **Palindrome String:**
   * **Input:** "racecar"
   * **Output:** "racecar"
4. **String with Special Characters:**
   * **Input:** "abc@123"
   * **Output:** "321@cba"

PROGRAMM:-

public class ReverseString {

public static void main(String[] args) {

String str = "hello"; // Test case 1: Normal String

// String str = ""; // Test case 2: Empty String

// String str = "racecar"; // Test case 3: Palindrome String

// String str = "abc@123"; // Test case 4: String with Special Characters

if (str.isEmpty()) {

System.out.println("The reversed string is: " + str);

return;

}

String reversedStr = reverseString(str);

System.out.println("The reversed string is: " + reversedStr);

}

public static String reverseString(String str) {

String reversedStr = "";

for (int i = str.length() - 1; i >= 0; i--) {

reversedStr += str.charAt(i);

}

return reversedStr;

}

}

**50.Calculate Fibonacci Series Up to N Terms**

**Scenario:**

Write a Java program to calculate and print the Fibonacci series up to n terms.

**Test Case Scenarios:**

1. **Positive Number of Terms:**
   * **Input:** N = 5
   * **Output:** 0 1 1 2 3
2. **Single Term:**
   * **Input:** N = 1
   * **Output:** 0
3. **Zero Terms:**
   * **Input:** N = 0
   * **Output:** The number of terms must be positive.
4. **Large Number of Terms (e.g., N = 10):**
   * **Input:** N = 10
   * **Output:** 0 1 1 2 3 5 8 13 21 34

PROGRAMM:-

import java.util.Scanner;

public class FibonacciSeries {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

int n = scanner.nextInt();

if (n <= 0) {

System.out.println("The number of terms must be positive.");

return;

}

int[] fibonacciSeries = calculateFibonacciSeries(n);

printFibonacciSeries(fibonacciSeries);

}

public static int[] calculateFibonacciSeries(int n) {

int[] fibonacciSeries = new int[n];

fibonacciSeries[0] = 0;

if (n == 1) {

return fibonacciSeries;

}

fibonacciSeries[1] = 1;

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

fibonacciSeries[i] = fibonacciSeries[i - 1] + fibonacciSeries[i - 2];

}

return fibonacciSeries;

}

public static void printFibonacciSeries(int[] fibonacciSeries) {

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

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

}

}

}

**51.Check for Palindrome**

**Scenario:**

Write a Java program to check if a given string is a palindrome.

**Test Case Scenarios:**

1. **Normal Palindrome:**
   * **Input:** "Madam"
   * **Output:** The string is a palindrome.
2. **Not a Palindrome:**
   * **Input:** "Hello"
   * **Output:** The string is not a palindrome.
3. **Palindrome with Special Characters:**
   * **Input:** "A man, a plan, a canal, Panama"
   * **Output:** The string is a palindrome.
4. **Empty String:**
   * **Input:** ""
   * **Output:** The string is a palindrome.

PROGRAMM:-

import java.util.Scanner;

public class PalindromeCheck {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

String str = scanner.nextLine();

if (isPalindrome(str)) {

System.out.println("The string is a palindrome.");

} else {

System.out.println("The string is not a palindrome.");

}

}

public static boolean isPalindrome(String str) {

str = str.replaceAll("[^a-zA-Z0-9]", "").toLowerCase(); // Remove special characters and convert to lowercase

int start = 0;

int end = str.length() - 1;

while (start < end) {

if (str.charAt(start)!= str.charAt(end)) {

return false;

}

start++;

end--;

}

return true;

}

}

**52.Sort an Array Using Bubble Sort**

**Scenario:**

Write a Java program to sort an array using the bubble sort algorithm.

**Test Case Scenarios:**

1. **Unsorted Array:**

o **Input:** Array: [5, 2, 9, 1, 5, 6]

* + **Output:** Sorted array: 1 2 5 5 6 9

1. **Already Sorted Array:**

o **Input:** Array: [1, 2, 3, 4, 5]

* + **Output:** Sorted array: 1 2 3 4 5

1. **Array with Duplicates:**

o **Input:** Array: [3, 5, 2, 2, 8]

* + **Output:** Sorted array: 2 2 3 5 8

1. **Single Element Array:**
   * **Input:** Array: [10]
   * **Output:** Sorted array: 10
2. **Empty Array:**
   * **Input:** Array: []
   * **Output:** Sorted array: (no output)

PROGRAMM:-

import java.util.Scanner;

public class BubbleSort {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the size of the array: ");

int n = scanner.nextInt();

int[] arr = new int[n];

System.out.println("Enter the elements of the array:");

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

arr[i] = scanner.nextInt();

}

bubbleSort(arr);

System.out.print("Sorted array: ");

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

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

}

}

public static void bubbleSort(int[] arr) {

int n = arr.length;

boolean swapped;

for (int i = 0; i < n - 1; i++) {

swapped = false;

for (int j = 0; j < n - i - 1; j++) {

if (arr[j] > arr[j + 1]) {

// Swap arr[j] and arr[j + 1]

int temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

swapped = true;

}

}

if (!swapped) {

break;

}

}

}

}

**53.Merge Two Sorted Arrays**

**Scenario:**

Write a Java program to merge two sorted arrays into a single sorted array.

**Test Case Scenarios:**

1. **Basic Merge:**
   * **Input:** array1 = {1, 3, 5}, array2 = {2,

4, 6}

* + **Output:** Merged array: [1, 2, 3, 4, 5, 6]

1. **Empty Array:**
   * **Input:** array1 = {}, array2 = {1, 2, 3}
   * **Output:** Merged array: [1, 2, 3]
2. **One Empty Array:**
   * **Input:** array1 = {1, 2, 3}, array2 = {}
   * **Output:** Merged array: [1, 2, 3]

PROGRAMM:-

import java.util.Scanner;

public class MergeSortedArrays {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the size of the first array: ");

int n1 = scanner.nextInt();

int[] array1 = new int[n1];

System.out.println("Enter the elements of the first array:");

for (int i = 0; i < n1; i++) {

array1[i] = scanner.nextInt();

}

System.out.print("Enter the size of the second array: ");

int n2 = scanner.nextInt();

int[] array2 = new int[n2];

System.out.println("Enter the elements of the second array:");

for (int i = 0; i < n2; i++) {

array2[i] = scanner.nextInt();

}

int[] mergedArray = mergeSortedArrays(array1, array2);

System.out.print("Merged array: ");

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

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

}

}

public static int[] mergeSortedArrays(int[] array1, int[] array2) {

int[] mergedArray = new int[array1.length + array2.length];

int i = 0, j = 0, k = 0;

while (i < array1.length && j < array2.length) {

if (array1[i] < array2[j]) {

mergedArray[k++] = array1[i++];

} else {

mergedArray[k++] = array2[j++];

}

}

while (i < array1.length) {

mergedArray[k++] = array1[i++];

}

while (j < array2.length) {

mergedArray[k++] = array2[j++];

}

return mergedArray;

}

}

**54.Implement Quick Sort**

**Scenario:**

Write a Java program to sort an array using the Quick Sort algorithm.

**Test Case Scenarios:**

1. **Basic Array:**

o **Input:** array = {10, 7, 8, 9, 1, 5}

* + **Output:** Sorted array: [1, 5, 7, 8, 9, 10]

1. **All Elements Same:**
   * **Input:** array = {5, 5, 5, 5}
   * **Output:** Sorted array: [5, 5, 5, 5]
2. **Reversed Array:**

o **Input:** array = {9, 8, 7, 6, 5}

* + **Output:** Sorted array: [5, 6, 7, 8, 9]

PROGRAMM:-

public class QuickSort {

public static void main(String[] args) {

int[] array = {10, 7, 8, 9, 1, 5};

quickSort(array, 0, array.length - 1);

System.out.print("Sorted array: ");

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

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

}

}

public static void quickSort(int[] array, int low, int high) {

if (low < high) {

int pi = partition(array, low, high);

quickSort(array, low, pi - 1);

quickSort(array, pi + 1, high);

}

}

public static int partition(int[] array, int low, int high) {

int pivot = array[high];

int i = (low - 1);

for (int j = low; j < high; j++) {

if (array[j] < pivot) {

i++;

int temp = array[i];

array[i] = array[j];

array[j] = temp;

}

}

int temp = array[i + 1];

array[i + 1] = array[high];

array[high] = temp;

return i + 1;

}

}

**43.Find All Prime Numbers Up to N**

**Scenario:**

Write a Java program to find all prime numbers up to a given number n using the Sieve of Eratosthenes algorithm.

**Test Case Scenarios:**

1. **Small Number:**

* **Input:** n = 10
* **Output:** Prime numbers up to 10: 2 3 5 7

1. **Large Number:**
   * **Input:** n = 50
   * **Output:** Prime numbers up to 50: 2 3 5 7 11

13 17 19 23 29 31 37 41 43 47

1. **No Prime Numbers:**
   * **Input:** n = 1
   * **Output:** There are no prime numbers less than 2.

PROGRAMM:-

import java.util.Scanner;

public class SieveOfEratosthenes {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

int n = scanner.nextInt();

boolean[] prime = new boolean[n + 1];

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

prime[i] = true;

}

prime[0] = prime[1] = false;

for (int p = 2; p \* p <= n; p++) {

if (prime[p]) {

for (int i = p \* p; i <= n; i += p) {

prime[i] = false;

}

}

}

System.out.print("Prime numbers up to " + n + ": ");

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

if (prime[i]) {

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

}

}

System.out.println();

}

}

**56.Find the Longest Common Prefix**

**Scenario:**

Write a Java program to find the longest common prefix string amongst an array of strings.

**Test Case Scenarios:**

1. **Common Prefix:**
   * **Input:** {"flower", "flow", "flight"}
   * **Output:** Longest common prefix: "fl"
2. **No Common Prefix:**
   * **Input:** {"dog", "racecar", "car"}
   * **Output:** Longest common prefix: ""
3. **All Identical Strings:**
   * **Input:** {"same", "same", "same"}
   * **Output:** Longest common prefix: "same"

PROGRAMM:-

public class LongestCommonPrefix {

public static void main(String[] args) {

String[] strs = {"flower", "flow", "flight"};

String longestCommonPrefix = findLongestCommonPrefix(strs);

System.out.println("Longest common prefix: " + longestCommonPrefix);

}

public static String findLongestCommonPrefix(String[] strs) {

if (strs == null || strs.length == 0) {

return "";

}

String prefix = strs[0];

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

while (strs[i].indexOf(prefix)!= 0) {

prefix = prefix.substring(0, prefix.length() - 1);

if (prefix.isEmpty()) {

return "";

}

}

}

return prefix;

}

}

57.**Find the Kth Largest Element in an Array**

**Scenario:**

Write a Java program to find the Kth largest element in an unsorted array.

**Test Case Scenarios:**

* **Input:**

o Array: [3, 2, 1, 5, 6, 4]

* + K: 2
* **Output:**
  + The 2nd largest element is: 5

1. *K Greater Than Array Size*
   * **Input:**
     + Array: [1, 2]
     + K: 3
   * **Output:** The 3rd largest element does not exist. (Program can be modified to handle this case by returning a specific value or throwing an exception.).
2. *K Equals 1*
   * **Input:**

o Array: [7, 10, 4, 3, 20, 15]

* + - K: 1
  + **Output:**
    - The 1st largest element is: 20

1. *All Elements Same*
   * **Input:**

o Array: [5, 5, 5, 5]

* + - K: 2
  + **Output:**
    - The 2nd largest element is: 5

1. *Array with Negative Numbers*
   * **Input:**

o Array: [-1, -3, -2, -5, -4]

* + - K: 3
  + **Output:**
  + The 3rd largest element is: -3

PROGRAMM:-

import java.util.Arrays;

public class KthLargestElement {

public static void main(String[] args) {

int[] nums = {3, 2, 1, 5, 6, 4};

int k = 2;

int kthLargest = findKthLargest(nums, k);

System.out.println("The " + k + "th largest element is: " + kthLargest);

}

public static int findKthLargest(int[] nums, int k) {

if (k > nums.length) {

System.out.println("The " + k + "th largest element does not exist.");

return Integer.MIN\_VALUE; // or throw an exception

}

Arrays.sort(nums);

return nums[nums.length - k];

}

}

**58.Basic Thread Creation**

*Question:*

Write a Java program to create and start two threads. Each thread should print a message indicating which thread it is, and the current thread ID. Use both Thread class and Runnable interface approaches.

*Test Case Scenarios:*

**General Case:**

* **Input:** No input required.
* **Expected Output**
  + **Thread is running: [Thread ID 1] Thread is running: [Thread ID 2]**

PROGRAMM:-

class MyThread extends Thread {

public void run() {

System.out.println("Thread is running: " + Thread.currentThread().getId());

}

}

class MyRunnable implements Runnable {

public void run() {

System.out.println("Thread is running: " + Thread.currentThread().getId());

}

}

public class Main {

public static void main(String[] args) {

// Using Thread class

MyThread thread1 = new MyThread();

MyThread thread2 = new MyThread();

thread1.start();

thread2.start();

// Using Runnable interface

MyRunnable runnable1 = new MyRunnable();

MyRunnable runnable2 = new MyRunnable();

Thread thread3 = new Thread(runnable1);

Thread thread4 = new Thread(runnable2);

thread3.start();

thread4.start();

}

}

**59.Thread Synchronization**

*Question:*

* + Write a Java program with a class Counter that has a synchronized method increment() to increment a count. Create two threads that each increment the count 1000 times. Print the final count value..

*Test Case Scenarios:*

**General Case:**

* **Input:** No input required.
* **Expected Output:**
  + Final count: 2000

PROGRAMM:-

Final count: 2000

public synchronized void increment() {

count++;

}

public int getCount() {

return count;

}

public CounterThread(Counter counter) {

this.counter = counter;

}

public void run() {

for (int i = 0; i < 1000; i++) {

counter.increment();

}

}

thread1.start();

thread2.start();

thread1.join();

thread2.join();

System.out.println("Final count: " + counter.getCount());

}

**60.Scheduled Tasks**

*Question:*

Write a Java program to use ScheduledExecutorService to schedule a task that prints the current time every 2 seconds. Schedule another task to shut down the scheduler after 10 seconds.

*Test Case Scenarios:*

**General Case:**

* **Input:** No input required.
* **Expected Output: Current time: [Time 1] Current time: [Time 2]**

**...**

* + **Scheduler shutdown.**

PROGRAMM:-

import java.time.LocalTime;

import java.util.concurrent.Executors;

import java.util.concurrent.ScheduledExecutorService;

import java.util.concurrent.ScheduledFuture;

import java.util.concurrent.TimeUnit;

public class Main {

public static void main(String[] args) {

ScheduledExecutorService scheduler = Executors.newScheduledThreadPool(2);

// Schedule a task to print the current time every 2 seconds

ScheduledFuture<?> timeTask = scheduler.scheduleAtFixedRate(() -> {

System.out.println("Current time: " + LocalTime.now());

}, 0, 2, TimeUnit.SECONDS);

// Schedule a task to shut down the scheduler after 10 seconds

scheduler.schedule(() -> {

System.out.println("Scheduler shutdown.");

scheduler.shutdown();

}, 10, TimeUnit.SECONDS);

}

}