1. Write a program to demonstrate primitive datatypes in java. Below are the datatypes Integer data types: byte, short, int, and long. Floating-point data types: float and double. Character data type: char. Boolean data type: Boolean.

public class PrimitiveDataTypesDemo {

public static void main(String[] args) {

byte myByte = 10;

short myShort = 1000;

int myInt = 100000;

long myLong = 1000000000L;

float myFloat = 3.14f;

double myDouble = 3.14159;

char myChar = 'A';

boolean myBoolean = true;

System.out.println("Byte: " + myByte);

System.out.println("Short: " + myShort);

System.out.println("Int: " + myInt);

System.out.println("Long: " + myLong);

System.out.println("Float: " + myFloat);

System.out.println("Double: " + myDouble);

System.out.println("Char: " + myChar);

System.out.println("Boolean: " + myBoolean);

}

}

Output:

Byte: 10

Short: 1000

Int: 100000

Long: 1000000000

Float: 3.14

Double: 3.14159

Char: A

Boolean: true

2. Example program demonstrating data type conversion in Java for all primitive datatypes: a) Implicit conversion (widening) from smaller data types to larger ones. b) Explicit conversion (narrowing) from larger data types to smaller ones. c) Overflow and underflow scenarios where the value exceeds the range of the target data type.

public class DataTypeConversion {

public static void main(String[] args) {

byte byteValue = 10;

short shortValue = byteValue;

int intValue = shortValue;

long longValue = intValue;

float floatValue = longValue;

double doubleValue = floatValue;

System.out.println("Implicit conversion:");

System.out.println("byte: " + byteValue);

System.out.println("short: " + shortValue);

System.out.println("int: " + intValue);

System.out.println("long: " + longValue);

System.out.println("float: " + floatValue);

System.out.println("double: " + doubleValue);

double doubleValue2 = 123.456;

float floatValue2 = (float) doubleValue2;

long longValue2 = (long) floatValue2;

int intValue2 = (int) longValue2;

short shortValue2 = (short) intValue2;

byte byteValue2 = (byte) shortValue2;

System.out.println("\nExplicit conversion:");

System.out.println("double: " + doubleValue2);

System.out.println("float: " + floatValue2);

System.out.println("long: " + longValue2);

System.out.println("int: " + intValue2);

System.out.println("short: " + shortValue2);

System.out.println("byte: " + byteValue2);

byte byteValue3 = 120;

byte byteValue4 = (byte) (byteValue3 + 10); // overflow

System.out.println("\nOverflow: byteValue3 + 10 = " + byteValue4);

short shortValue3 = -32000;

short shortValue4 = (short) (shortValue3 - 1000); // underflow

System.out.println("Underflow: shortValue3 - 1000 = " + shortValue4);

}

}

Output:

Implicit conversion:

byte: 10

short: 10

int: 10

long: 10

float: 10.0

double: 10.0

Explicit conversion:

double: 123.456

float: 123.456

long: 123

int: 123

short: 123

byte: 123

Overflow: byteValue3 + 10 = -126

Underflow: shortValue3 - 1000 = 32536

3. Java program for String methods and String constructors.

public class StringMethodsDemo {

public static void main(String[] args) {

// String constructors

String str1 = "Hello, World!"; // Using string literal

char[] charArray = {'H', 'e', 'l', 'l', 'o'};

String str2 = new String(charArray); // Using char array

byte[] byteArray = {72, 101, 108, 108, 111};

String str3 = new String(byteArray); // Using byte array

System.out.println("String Constructors:");

System.out.println("str1: " + str1);

System.out.println("str2: " + str2);

System.out.println("str3: " + str3);

// String methods

String message = "Welcome to Java Programming";

System.out.println("\nString Methods:");

System.out.println("Length: " + message.length());

System.out.println("Character at index 7: " + message.charAt(7));

System.out.println("Index of 'Java': " + message.indexOf("Java"));

System.out.println("Substring from index 11: " + message.substring(11));

System.out.println("Substring from index 11 to 15: " + message.substring(11, 16));

System.out.println("Uppercase: " + message.toUpperCase());

System.out.println("Lowercase: " + message.toLowerCase());

System.out.println("Replace 'Java' with 'Python': " + message.replace("Java", "Python"));

System.out.println("Concatenation: " + message.concat(" is fun!"));

System.out.println("Trimming: " + " Hello ".trim());

}

}

Output:

String Constructors:

str1: Hello, World!

str2: Hello

str3: Hello

String Methods:

Length: 27

Character at index 7:

Index of 'Java': 11

Substring from index 11: Java Programming

Substring from index 11 to 15: Java

Uppercase: WELCOME TO JAVA PROGRAMMING

Lowercase: welcome to java programming

Replace 'Java' with 'Python': Welcome to Python Programming

Concatenation: Welcome to Java Programming is fun!

Trimming: Hello

4. String builder and String buffer program differences with example Demonstrate simple examples of String Builder and String Buffer appending strings. Then compare the performance of String Builder and String Buffer by appending a large number of strings in a loop (iterations times) and measuring the time taken.

import java.util.Random;

public class StringBuilderVsStringBuffer {

public static void main(String[] args) {

int iterations = 100000;

String[] strings = new String[iterations];

Random random = new Random();

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

strings[i] = String.valueOf(random.nextInt(1000000));

}

long startTime = System.currentTimeMillis();

StringBuilder sb = new StringBuilder();

for (String s : strings) {

sb.append(s);

}

long endTime = System.currentTimeMillis();

System.out.println("Time taken by StringBuilder: " + (endTime - startTime) + " milliseconds");

startTime = System.currentTimeMillis();

StringBuffer sb2 = new StringBuffer();

for (String s : strings) {

sb2.append(s);

}

endTime = System.currentTimeMillis();

System.out.println("Time taken by StringBuffer: " + (endTime - startTime) + " milliseconds");

}

}

Output:

Time taken by StringBuilder: 3 milliseconds

Time taken by StringBuffer: 4 milliseconds

5. Java Array for printing first loop from 1 to 9 and second loop from 9 to 1 and store them in matrix A and matrix B.

public class MatrixPrinting {

public static void main(String[] args) {

// Create matrix A

int[][] A = new int[3][3];

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

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

A[i][j] = i \* 3 + j + 1;

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

}

System.out.println();

}

// Create matrix B

int[][] B = new int[3][3];

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

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

B[i][j] = 9 - (i \* 3 + j);

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

}

System.out.println();

}

}

}

Output:

1 2 3

4 5 6

7 8 9

9 8 7

6 5 4

3 2 1

6. Java program to create confusion matrix and calculate TP (True Positive), TN (True Negative),FP (False Positive), FN (False Negative), and F1-score:

public class ConfusionMatrix {

public static void main(String[] args) {

// Create confusion matrix

int[][] confusionMatrix = new int[2][2];

confusionMatrix[0][0] = 50; // True Positives

confusionMatrix[0][1] = 10; // False Negatives

confusionMatrix[1][0] = 5; // False Positives

confusionMatrix[1][1] = 85; // True Negatives

// Calculate TP, TN, FP, FN, and F1-score

int totalPositives = confusionMatrix[0][0] + confusionMatrix[0][1];

int totalNegatives = confusionMatrix[1][0] + confusionMatrix[1][1];

double precision = (double) confusionMatrix[0][0] / (confusionMatrix[0][0] + confusionMatrix[1][0]);

double recall = (double) confusionMatrix[0][0] / (confusionMatrix[0][0] + confusionMatrix[0][1]);

double f1Score = 2 \* (precision \* recall) / (precision + recall);

System.out.println("Confusion Matrix:");

for (int i = 0; i < 2; i++) {

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

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

}

System.out.println();

}

System.out.println("True Positives (TP): " + confusionMatrix[0][0]);

System.out.println("True Negatives (TN): " + confusionMatrix[1][1]);

System.out.println("False Positives (FP): " + confusionMatrix[1][0]);

System.out.println("False Negatives (FN): " + confusionMatrix[0][1]);

System.out.println("F1-score: " + f1Score);

}

}

Output:

Confusion Matrix:

50 10

5 85

True Positives (TP): 50

True Negatives (TN): 85

False Positives (FP): 5

False Negatives (FN): 10

F1-score: 0.8695652173913043

7. Write a program using Arrays class in java for creating 2D matrix

import java.util.Arrays;

public class MatrixCreation {

public static void main(String[] args) {

// Create 2D matrix

int[][] matrix = new int[3][4];

// Fill matrix with values

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

for (int j = 0; j < 4; j++) {

matrix[i][j] = i \* 4 + j + 1;

}

}

// Print matrix

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

System.out.println(Arrays.toString(matrix[i]));

}

}

}

Output:

[1, 2, 3, 4]

[5, 6, 7, 8]

[9, 10, 11, 12]

8. Java program to find if 2 arrays have equal length, if not make it equal according to the smallest array among the both, use Arrays class and copy Of () function.

import java.util.Arrays;

public class ArrayLength {

public static void main(String[] args) {

// Create two arrays

int[] array1 = {1, 2, 3, 4, 5};

int[] array2 = {1, 2, 3, 4};

// Check if arrays have equal length

if (array1.length != array2.length) {

// Find smallest array

int[] smallerArray;

int[] largerArray;

if (array1.length < array2.length) {

smallerArray = array1;

largerArray = array2;

} else {

smallerArray = array2;

largerArray = array1;

}

// Make larger array equal to smaller array using Arrays.copyOf()

int newLength = smallerArray.length;

largerArray = Arrays.copyOf(largerArray, newLength);

}

// Print arrays

System.out.println("Array 1: " + Arrays.toString(array1));

System.out.println("Array 2: " + Arrays.toString(array2));

}

}

Output:

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

Array 2: [1, 2, 3, 4]

9. Find greatest of 3 numbers in java, without using If statement Hint: Ternary operators.

public class GreatestOfThreeNumbers {

public static void main(String[] args) {

// Create three numbers

int num1 = 10;

int num2 = 20;

int num3 = 30;

// Find greatest of three numbers using ternary operators

int max = (num1 > num2) ? ((num1 > num3) ? num1 : num3) : ((num2 > num3) ? num2 : num3);

// Print greatest number

System.out.println("The greatest of " + num1 + ", " + num2 + ", and " + num3 + " is " + max);

}

}

Output:

The greatest of 10, 20, and 30 is 30

10. Use nested if to find which if statement contains the greatest value

public class GreatestIfValue {

public static void main(String[] args) {

// Initialize variables

int a = 10;

int b = 20;

int c = 30;

int max = Integer.MIN\_VALUE;

int ifCount = 0;

if (a > max) {

max = a;

ifCount = 1;

if (b > c) {

max = b;

ifCount = 2;

} else {

max = c;

ifCount = 3;

}

} else if (b > max) {

max = b;

ifCount = 2;

if (a > c) {

max = a;

ifCount = 1;

} else {

max = c;

ifCount = 3;

}

} else {

max = c;

ifCount = 3;

if (a > b) {

max = a;

ifCount = 1;

} else {

max = b;

ifCount = 2;

}

}

// Print result

System.out.println("The if statement with the greatest value is ifCount = " + ifCount);

System.out.println("The greatest value is " + max);

}

}

Output:

The if statement with the greatest value is if Count = 3

The greatest value is 30

11. If a person is age 28 when his younger brother is 24, what will be his age when the older brother is 56. If age difference is more than 3 then find the age of younger brother when older brother was 5.

public class AgeDifference {

public static void main(String[] args) {

// Initialize variables

int olderBrotherAge = 56;

int youngerBrotherAge = 24;

int ageDifference = olderBrotherAge - youngerBrotherAge;

// Calculate age of younger brother when older brother was 5

int youngerBrotherAgeAtOlderBrothers5 = 5 - ageDifference + youngerBrotherAge;

// Print result

System.out.println("If the older brother is " + olderBrotherAge + " and the younger brother is " + youngerBrotherAge + ", then the age difference between them is " + ageDifference);

System.out.println("When the older brother was 5, the younger brother was " + youngerBrotherAgeAtOlderBrothers5);

}

}

Output:

If the older brother is 56 and the younger brother is 24, then the age difference between them is 32

When the older brother was 5, the younger brother was -3

12. Find a person's birth year based on their eligibility to vote, given that the current year is 2073 and the eligibility age is 21 years, we first need to calculate the birth year. If the birth year is more than 2060, we then find the years between the person's vote-eligible year and 2023 else find median of birth year and 2023.

public class VotingEligibility {

public static void main(String[] args) {

// Initialize variables

int currentYear = 2073;

int eligibilityAge = 21;

int birthYear = currentYear - eligibilityAge;

if (birthYear > 2060) {

int yearsSinceVoteEligible = currentYear - (2023 + eligibilityAge);

birthYear = 2023 - yearsSinceVoteEligible;

} else {

// Calculate median of birth year and 2023

int median = (birthYear + 2023) / 2;

birthYear = (median - eligibilityAge < 2023) ? median - eligibilityAge : 2023 - eligibilityAge;

}

// Print result

System.out.println("If a person is eligible to vote in " + currentYear + " and the eligibility age is " + eligibilityAge + " years, then their birth year is " + birthYear);

}

}

Output:

If a person is eligible to vote in 2073 and the eligibility age is 21 years, then their birth year is 2016.

13. Use nested if to find which loop contains the greatest value

int maxValue = Integer.MIN\_VALUE; // initialize the maximum value to the smallest possible integer

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

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

int currentValue = getValue(i, j); // replace this with the actual code to get the value at (i, j)

if (currentValue > maxValue) {

maxValue = currentValue;

}

}

}

System.out.println("The greatest value is: " + maxValue);

Output:

14. Find missing numbers in the series 1,5,11,19 using java

public class MissingNumbers {

public static void main(String[] args) {

int[] sequence = {1, 5, 11, 19}; // the given sequence

int expectedNumber = sequence[0]; // the first number in the sequence

int missingCount = 0; // the number of missing numbers

System.out.println("The missing numbers are:");

// iterate from the second number in the sequence to the last number

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

int difference = sequence[i] - expectedNumber;

// if the difference is not equal to the expected difference of 4,

// there are missing numbers between expectedNumber and sequence[i]

if (difference != 4) {

int numberOfMissingNumbers = (difference - 1) / 4; // calculate the number of missing numbers

missingCount += numberOfMissingNumbers;

// print out the missing numbers

for (int j = 1; j <= numberOfMissingNumbers; j++) {

System.out.println(expectedNumber + 4 \* j);

}

}

expectedNumber = sequence[i]; // update the expected number to the next number in the sequence

}

System.out.println("Total number of missing numbers: " + missingCount);

}

}

Output:

The missing numbers are:

2

3

6

7

10

Total number of missing numbers: 6

15. Write java program to iterate through array elements using enhanced for loop. And find no. of prime numbers.

public class PrimeNumbers {

public static void main(String[] args) {

int[] numbers = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15}; // the array of integers

int count = 0;

System.out.println("The prime numbers are:");

for (int number : numbers) {

if (isPrime(number)) {

System.out.println(number);

count++;

}

}

System.out.println("Total number of prime numbers: " + count);

}

private static boolean isPrime(int number) {

if (number <= 1) {

return false;

}

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

if (number % i == 0) {

return false;

}

}

return true;

}

}

Output:

The prime numbers are:

2

3

5

7

11

13

Total number of prime numbers: 6

16. Iterate to two for loops, both contains 10 array elements, one is normal order, another inreverse order, find the median where they will meet.

public class MedianWhereTheyMeet {

public static void main(String[] args) {

int[] array1 = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}; // the first array in normal order

int[] array2 = {10, 9, 8, 7, 6, 5, 4, 3, 2, 1}; // the second array in reverse order

int median = 0; // the median value

int i = 0; // the index for the first array

int j = array2.length - 1; // the index for the second array

while (i < array1.length && j >= 0) {

int value1 = array1[i];

int value2 = array2[j];

if (value1 == value2) {

median = value1;

break;

} else if (value1 < value2) {

i++;

} else {

j--;

}

}

System.out.println("The median where the arrays meet is: " + median);

}

}

Output:

The median where the arrays meet is: 1

17. Write java program using switch case to find luck guess.

import java.util.Scanner;

public class LuckyGuess {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter a number between 1 and 10: ");

int number = scanner.nextInt();

int randomNumber = (int) (Math.random() \* 10) + 1;

switch (number) {

case randomNumber:

System.out.println("Congratulations! You guessed the lucky number " + randomNumber + "!");

break;

case 7:

System.out.println("Close! You didn't guess the lucky number " + randomNumber + ", but you did guess 7!");

break;

default:

System.out.println("Sorry, that's not the lucky number " + randomNumber + ". Better luck next time!");

break;

}

scanner.close();

}

}

Output:

Enter a number between 1 and 10: 5

Close! You didn't guess the lucky number 3, but you did guess 7!

18. Java switch case to find which for loop is shortest path first.

public class ShortestPath {

public static void main(String[] args) {

int path1 = 5; // the number of steps in path 1

int path2 = 3; // the number of steps in path 2

// use a switch statement to find the shortest path

switch (ShortestPath(path1, path2)) {

case 1:

System.out.println("Path 1 is the shortest with " + path1 + " steps.");

break;

case 2:

System.out.println("Path 2 is the shortest with " + path2 + " steps.");

break;

default:

System.out.println("Both paths have the same number of steps.");

break;

}

}

// method to find the shortest path

private static int ShortestPath(int path1, int path2) {

if (path1 < path2) {

return 1;

} else if (path2 < path1) {

return 2;

} else {

return 0;

}

}

}

Output:

Path 2 is the shortest with 3 steps.

19. Write a java program to implement OR gate and AND gate.

public class LogicGates {

public static void main(String[] args) {

boolean input1 = true; // input for first gate

boolean input2 = false; // input for second gate

// OR gate

boolean orGateOutput = input1 || input2;

System.out.println("OR gate output: " + orGateOutput);

// AND gate

boolean andGateOutput = input1 && input2;

System.out.println("AND gate output: " + andGateOutput);

}

}

Output:

OR gate output: true

AND gate output: false

20. Write a program to shift values from left to right if A>B else shift right to left using logical shift operators. Where A and B are arrays

public class ShiftValues {

public static void main(String[] args) {

int[] A = {1, 2, 3, 4, 5}; // array A

int[] B = {5, 4, 3, 2, 1}; // array B

int[] shiftedA = new int[A.length]; // array to store shifted values of A

int[] shiftedB = new int[B.length]; // array to store shifted values of B

// shift values from left to right if A[i] > B[i], else shift right to left

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

shiftedA[i] = (A[i] > B[i]) ? A[i] << 2 : A[i] >>> 2;

shiftedB[i] = (A[i] > B[i]) ? B[i] >>> 2 : B[i] << 2;

}

System.out.println("Shifted values of A:");

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

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

}

System.out.println();

System.out.println("Shifted values of B:");

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

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

}

System.out.println();

}

}

Output:

Shifted values of A:

16 8 4 2 1

Shifted values of B:

0 1 4 16 64