Java\_assignment\_04

## Prompt for the input

Prompt for the input numbers based on the operation and do the operation

C:\> java coe\_operate

Enter the operation (ADD/SUB/PROD/DIV): *user enters* ‘ADD’

Enter the 1st number: *user enters* 2

Enter the 2nd number: *user enters* 5

**Output:** The addition of 2 numbers 2 & 5 is 7

*public class JavaProgram*

*{*

*public static void main(String args[])*

*{*

*int a, b, res;*

*Scanner scan = new Scanner(System.in);*

*System.out.print("Enter Two Numbers : ");*

*a = scan.nextInt();*

*b = scan.nextInt();*

*res = a + b;*

*System.out.println("Addition = " +res);*

*res = a - b;*

*System.out.println("Subtraction = " +res);*

*res = a \* b;*

*System.out.println("Multiplication = " +res);*

*res = a / b;*

*System.out.println("Division = " +res);*

*}*

*}*

## Scan and Ifs

Sort 3 numbers using only if condition. Numbers should be retrieved from the user

C:\> java coe\_sort 3 7 2

The numbers sorted in ascending order are 2 3 & 7

The numbers sorted in descending order are 7 3 & 2

Input : arr[] = {3,7,2}

Output : arr[] = {2,3,7}

Output : arr[] = {7,3,2}

void twoWaySort(int arr[], int n)

{

// Current indexes from left and right

int l = 0, r = n-1;

// Count of odd numbers

int k = 0;

while (l < r)

{

// Find first odd number from left side.

while (arr[l]%2 != 0)

{

l++;

k++;

}

// Find first even number from right side.

while (arr[r]%2 == 0 && l<r)

r--;

// Swap odd number present on left and even

// number right.

if (l < r)

swap(arr[l], arr[r]);

}

// Sort odd number in descending order

sort(arr, arr+k, greater<int>());

// Sort even number in ascending order

sort(arr+k, arr+n);

}

// Driver code

int main()

{

int arr[] = {3,7,2};

int n = sizeof(arr)/sizeof(int);

twoWaySort(arr, n);

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

cout << arr[i] << " ";

return 0;

}

3. Solve a 2-dimensional equation

Write a program to solve a two dimensional equations:

e.g:-

nX+mY=c1;

aX+bY=c2;

Input:

Enter 1st equation’s n:

Enter 1st equation’s m:

Enter 1st equation’s constant:

Enter 2nd equation’s a:

Enter 2nd equation’s b:

Enter 2nd equation’s constant:

Output:

The given Equation is:

nX+mY=c1;

aX+bY=c2;

public static void solveSimultaneousEquations(double a, double b, double c, double d, double e, double f) {

double det = 1/ ((a) \* (d) - (b) \* (c));

double x = ((d) \* (e) - (b) \* (f)) / det;

double y = ((a) \* (f) - (c) \* (e)) / det;

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

}

public static void solveSimultaneousEquations(double a, double b, double c, double d, double e, double f) {

double det = ((a) \* (d) - (b) \* (c)); //instead of 1/

double x = ((d) \* (e) - (b) \* (f)) / det;

double y = ((a) \* (f) - (c) \* (e)) / det;

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

}

## Chessboard

Write a program to display a chess board as below

1. Consider x as black O as white
2. If the input is 1x1 matrix the black and white pattern should be X and O
3. If the input is 2x1 matrix the black and white pattern shoud be XX and OO
4. Likewise the 'X' and 'O' pattern should be designed using runtime value
5. Use '|' as vertical separator.
6. Use '-' as horizontal separator.
7. The size of the chess board and size of Pattern will be retrieved from the user.

**Sample output:**

// Java implementation of the approach

class GFG

{

// Function that returns true if white wins

static boolean whiteWins(int rowW, int colW,

int rowB, int colB)

{

int white = 0, black = 0;

boolean flag=true;

while (flag)

{

// If white can move

if (rowW != 8)

{

// If white pawn can kill black pawn

// White wins

if (rowB == rowW + 1

&& (colB == colW - 1 || colB == colW + 1))

return true;

// Make the move forward

else

rowW++;

}

// White has no moves

// White loses

else

return false;

// If black can move

if (rowB != 1)

{

// If black pawn can kill white pawn

// White loses

if (rowB == rowW + 1

&& (colB == colW - 1 || colB == colW + 1))

return false;

// Make the move forward

else

rowB--;

}

// Black has no moves

// White wins

else

return true;

}

// If white has got more moves

if (white > black)

return true;

return false;

}

// Driver code

public static void main(String args[])

{

int rowW = 2, colW = 2, rowB = 3, colB = 3;

if (whiteWins(rowW, colW, rowB, colB))

System.out.println("White");

else

System.out.println("Black");

}

}

// Java implementation of above approach

class GFG

{

// Function to return the number

// of squares that the king can

// reach in the given number of moves

static int Square(int row, int column,

int moves)

{

int a = 0, b = 0, c = 0,

d = 0, total = 0;

// Calculate initial and final coordinates

a = row - moves;

b = row + moves;

c = column - moves;

d = column + moves;

// Since chessboard is of size 8X8

// so if any coordinate is less

// than 1 or greater than 8 make

// it 1 or 8.

if (a < 1)

a = 1;

if (c < 1)

c = 1;

if (b > 8)

b = 8;

if (d > 8)

d = 8;

// Calculate total positions

total = (b - a + 1) \* (d - c + 1) - 1;

return total;

}

// Driver code

public static void main(String []args)

{

int R = 4, C = 5, M = 2;

System.out.println(Square(R, C, M));

}

}

// This code is contributed by Ita\_c.

import java.io.\*;

class GFG

{

// Function to return the

// value of ncr effectively

static int ncr(int n, int r)

{

// Initialize the answer

int ans = 1;

for (int i = 1; i <= r; i += 1)

{

// Divide simultaneously by

// i to avoid overflow

ans \*= (n - r + i);

ans /= i;

}

return ans;

}

// Function to return the count of required ways

static int totalWays(int X, int Y, int M, int W)

{

return (ncr(M, X) \* ncr(W, Y));

}

// Driver code

public static void main (String[] args)

{

int X = 4, Y = 3, M = 6, W = 5;

System.out.println(totalWays(X, Y, M, W));

}

}

|  |
| --- |
| // Java program to find the number of ways  // Calculate total ways to place 'x' and 'y'  // at n places such that no two 'x' are together  public class GFG {        // Function to return number of ways      static int ways(int n)      {          // for n=1          int first = 2;            // for n=2          int second = 3;          int res = 0;            // iterate to find Fibonacci term          for (int i = 3; i <= n; i++) {              res = first + second;              first = second;              second = res;          }            return res;      }      public static void main(String[] args)      {            // total number of places          int n = 7;            System.out.print("Total ways are: " + ways(n));      }  } |