Lab 6: Loops #2

**Instructions:** Please structure the lab with the main method providing a menu for the other methods as you did in previous labs.

public class Lab6 {

public static void main(String[] args) {

// Menu system here

//if they selected problem 1

problem1();

//if they selected problem 2

problem2();

//...

}

public static void problem1(){

//Code for problem 1 here.

}

public static void problem2(){

//Code for problem 2 here.

}

}

| Requirement |
| --- |
|  |
| Lab compiles |
| All programs run without errors. |

1. Write a calculator method that displays a menu of operations and allows the user to perform simple calculations. Make sure your calculator can handle the operators +, -, \* and /. This method should:
   * Prompt the user to let them know what the method/program does.
   * Wait for input for a number.
   * Wait for input for an operator.
   * Wait for input for another number.
   * Print the result.
   * Program should not error with invalid calculations.

**Ensure that the user can enter ints or doubles.**

Tip: If you are prompted for input you can put more than one thing on the line separated by spaces and if you are not using .nextLine() it will count as multiple inputs.

**Example:**

Please enter a number followed by an operator (+, -, \* or /) and then another number.

* 5
* \*
* 10

Result: 50

**OR**

Please enter a number followed by an operator (+, -, \* or /) and then another number.

* 5 \* 10

Result: 50

| Requirement |
| --- |
| Meaningfully prompts user and validates input. |
| Produces correct output for + operator |
| Produces correct output for - operator |
| Produces correct output for \* operator |
| Produces correct output for / operator |
| Validates data for Divide by 0 errors. |

1. Write a method using nested for loops that produces the following output:

1 2 3 4 5

2 3 4 5 6

3 4 5 6 7

4 5 6 7 8

5 6 7 8 9

| Requirement |
| --- |
| Uses nested loops |
| Calculates values in loops and does not just print out the values. |
| Produces correct output. |

1. Write a method that uses a 2d array to keep track of the position of a piece on a board as it moves. The 2d array should be a 9 by 9 matrix with the piece starting in the center of the board. The program should begin printing the board showing the piece in the middle:

- - - - - - - - -

- - - - - - - - -

- - - - - - - - -

- - - - - - - - -

- - - - ■ - - - -

- - - - - - - - -

- - - - - - - - -

- - - - - - - - -

- - - - - - - - -

(I used (char)254 for the piece, you can use whatever you want to represent it.)

The method should then prompt the user to make any number of moves using N, S, E, W and update the position of the piece in the 2d array. N will move the piece up on the board, S down, E to the right, and W to the left.

If the piece would go off of the board, it should appear on the other side of the board as if it was ‘wrapping’ around the board. So if the piece was at the top and the user types N to go up more, it should appear at the bottom in the same column.

Example:

- - - - - - - - -

- - - - - - - - -

- - - - - - - - -

- - - - - - - - -

- - - - ■ - - - -

- - - - - - - - -

- - - - - - - - -

- - - - - - - - -

- - - - - - - - -

Enter the direction you would like to move, or type Quit to end:

N

Enter the number of spaces you would like to move N:

3

- - - - - - - - -

- - - - ■ - - - -

- - - - - - - - -

- - - - - - - - -

- - - - - - - - -

- - - - - - - - -

- - - - - - - - -

- - - - - - - - -

- - - - - - - - -

You do not need to print out the board between moves. The method ends when the user enters Quit, and the board with the new piece location is then printed.

| Requirement |
| --- |
| Board printed using loops. |
| Board stored in 9x9 multi-dimensional array. |
| Validates user input. |
| Uses the quit string as a sentinel value. |
| Properly handles board ‘wrapping’ around the sides |
| Produces correct output. |