Lab 12 – FindTheSums

Introduction

"Word Search" puzzles are popular games where players are given a 2D (two dimensional) array of letters and the goal is to find words that are spelled horizontally, vertically, and diagonally. In this lab, we will do something similar, but we will use integers and sums instead of letters and words. We'll find horizontal and vertical sums in a 2D input array of integers that equal some input integer value (i.e. find all horizontal sums in a 2D array that equal 20). We won't be finding diagonal sums in this lab, but feel free to challenge yourself with diagonal sums after you've submitted your lab. This lab will sharpen your problem solving skills and give you hands-on experience programming with 2D arrays. It is important to note that when you are working with 2D arrays, you'll need to use nested loops to iterate through the values in their rows and columns.

For this lab, you'll be working with 2D input arrays of integers that have **m** rows and **n** columns, where **m** > 0 and **n** > 0, and the input arrays contain only integers ranging from 1 to 9 (inclusive). The goals of this lab are to write a method that convert a 2D array to a neatly printable String and to write two additional methods that find the horizontal and vertical sums for a 2D input array and an input integer called **sumToFind**. For example, if **sumToFind** is 20, then your horizontal sum method would find all horizontally adjacent values in the input array that equal 20 and put them into a new output array, and values that aren't in a horizontal sum equal to 20 would be set to zero in the output array. Similarly, vertical sums will be found the same way except their sums will be vertical. Study the examples in Figure 1. Please note that sums may overlap as shown in the highlighted example in Figure 1.

Figure 1

Horizontal Sums sumToFind = 20	Input Array	Vertical Sums sumToFind = 20
0 0 0 0 0 0 0 0 0 0	7 3 8 5 6 7 4 1 9 5	0 0 0 0 6 0 0 1 0 5
0 1 6 1 8 4 0 0 0 0	8 1 6 1 8 4 6 9 9 6	8 1 0 0 8 0 0 9 0 6
0 2 4 8 6 1 1 0 0 0	9 2 4 8 6 1 1 3 6 2	9 2 0 0 6 0 1 3 0 2
3 6 8 3 0 0 0 0 0 0	3 6 8 3 1 9 2 7 9 6	3 6 8 0 1 9 2 7 9 6
0 7 7 6 3 5 6 4 2 0	5 7 7 6 3 5 6 4 2 1	0 7 7 0 3 5 6 4 2 1
6 4 5 5 0 0 0 0 0 0	6 4 5 5 7 6 8 1 9 7	6 4 5 0 7 6 8 1 9 0
0 0 5 4 3 7 1 0 0 0	8 4 5 4 3 7 1 2 1 8	8 4 0 0 3 0 1 2 1 0
6 8 6 0 8 6 2 4 6 2	6 8 6 7 8 6 2 4 6 2	6 8 0 0 8 0 2 4 6 0
0 0 0 0 0 8 2 2 8 0	7 8 6 8 3 8 2 2 8 5	0 8 0 0 3 0 0 2 8 0
0 7 7 6 0 2 9 9 0 0	8 7 7 6 6 2 9 9 5 8	0 0 0 0 6 0 0 0 5 0
Note: 8 6 2 4 6 2 contains two overlapping sums that equal 20: 8 + 6 + 2 + 4 and 6 + 2 + 4 + 6 + 2		

In this lab, you will create a class called **`FindTheSums`** that has the following public static methods: **`arrayToString`**, **`horizontalSums`**, and **`verticalSums`**.

Lab Objectives

By the end of the lab, you should have gained experience working with 2D arrays, nested loops, nested statements, static methods, writing pseudocode, and problem decomposition.

Prerequisites

The main concepts in this lab are from Chapter 7. Concepts from Chapters 4, 5, and 6 are also in this lab.

Lab 12 – FindTheSums

What to Submit

The **`FindTheSums.java`** file should be submitted to eLC for grading.

Instructions

1. Create a class called **`FindTheSums`**. Include the standard header comment stating your name, the date, program purpose, and containing the statement of academic honesty. When writing, you must also abide by the Java formatting and style conventions.
2. Study the examples in Figure 1 to understand how horizontal and vertical sums are found. This lab requires more time thinking about how to solve the problems than previous labs. Experienced programmers spend more time thinking about how to solve a problem than implementing its solution. To start, use your fingers to slowly trace through the values in the input array in Figure 1 one-by-one from left to right and top to bottom to find the horizontal and vertical sums that equal 20. As you do this, ask yourself how many loops and variables are needed and what strategies would allow you to find all of the horizontal and vertical sums without missing any of them. These strategies are what you'll be implementing in Java with two methods: a method to find the horizontal sums and another method to find the vertical sums. Write out the logic of your methods in pseudocode on a piece of paper (you may find it useful to look at the method definitions in the next step when writing your pseudocode). This will help you decompose the problems into simpler parts that will make writing the methods in the next step easier.
3. In the class, you should implement the methods below, and what these methods return should match the examples at end of this lab.
    a. **`public static String arrayToString(int[][] a)`**
    This method will return a String that is a neat representation of the values in **`a`**. By neat, we mean that values in each column of **`a`** have a single space between them and the rows have a single newline character between them. There should not be a space before the first value in a column or after the last value in a column. Also, there should not be a newline before the first row or after the last row.
    b. **`public static int[][] horizontalSums(int[][] a, int sumToFind)`**
    This method will create a new output array called **`b`** that has the same dimensions as **`a`**. For each **`a`**[i][j], where i and j are valid indices in **`a`**, if **`a`**[i][j] is part of a horizontal sum in **`a`** that equals **`sumToFind`**, then **`b`**[i][j] = **`a`**[i][j]; otherwise, **`b`**[i][j] = 0. The method should return **`b`**.
    c. **`public static int[][] verticalSums(int[][] a, int sumToFind)`**
    This method will create a new output array called **`b`** that has the same dimensions as **`a`**. For each **`a`**[i][j], where i and j are valid indices in **`a`**, if **`a`**[i][j] is part of a vertical sum in **`a`** that equals **`sumToFind`**, then **`b`**[i][j] = **`a`**[i][j]; otherwise, **`b`**[i][j] = 0. The method should return **`b`**.
4. Download the **`FindTheSumsTester.java`** file, and place it in the same directory as your class. Your class must compile correctly with **`FindTheSumsTester.java`**. Run **`FindTheSumsTester`** to test your methods, and verify that your output matches the output at the end of this lab. If the output differs, then you have bugs that must be fixed. You should also create additional tests in **`FindTheSumsTester`** to further test your methods. Your methods must work for any valid inputs.

In your code, you are not allowed to use any java.util.Arrays methods or the java stream API (if you are familiar with either of these). Using the java.util.Arrays class or Java stream in any way will result in a grade of zero on this assignment.

Additional Requirements

These are things that make the graders lives easier, and ultimately, you will see in the real world as well. Remember the teaching staff does not want to touch your code after they gave you requirements; they want to see the perfect results they asked for! Here is a checklist of things you can **lose points** for:

- (-1 point) If the source file(s)/class(es) are named incorrectly (case matters!)
- (-1 point) If your source file(s) have a package declaration at the top
- (-1 point) If any source file you submit is missing your Statement of Academic Honesty at the top of the source file. All submitted source code files must contain your Statement of Academic Honesty at the top of each file.
- (-1 point) If you have more than one instance of Scanner in your program.
- (-1 point) Inconsistent I/O (input/output) that does not match our instructions or examples exactly (unless otherwise stated in an assignment's instructions). Your program's I/O (order, wording, formatting, etc.) must match our examples and instructions.
- (-7 points) If the source file(s) are not submitted before this assignment's specified deadline. Late submissions will not be accepted for this lab since it is the last lab of the semester.
- (-7 points) If the source file(s) do not compile. All classes must be declared as public and all method signatures must match exactly what is in these instructions. Your program must compile on the command line using the `javac` command and using the required Java compiler for this course. Students are responsible for checking that their source code file(s) compile using the `javac` command for all methods in these instructions.
- If your (-1 point) comments or (-1 point) variables are "lacking"
    - Here, "lacking" means that you or a TA can find **any** lines of code or variables that take more than 10 seconds to understand, and there is no comment, or the variable name does not make sense (variable names like **b**, **bb**, **bbb**, etc. **will almost never be acceptable)**
- (-1 point) Indentation is not consistent throughout your source code
    - Refresh your memory of indentation patterns in chapter 2 in the course textbook
    - Be careful of a combination of tabs and spaces in your files (use one or the other)
- (-7 points) If you use a non-standard Java library or class (StringBuilder class, Arrays class, any class in java.util.stream) or other code specifically disallowed by the lab/project.

If any of the above do not make sense to you, then talk to a TA or your lab instructor.

eLC Submission and Grading

After you have completed and thoroughly tested your program, upload **`FindTheSums.java`** to ***eLC*** to receive credit. Always double check that your submission was successful on ***eLC***!

- Students may earn 3 points for lab attendance and 7 points if their program passes various test cases that we use for grading. **Some test cases may use values taken from the examples in this document and some test cases may not**. You should come up with additional test cases to check

that your program is bug free. Your submitted program will be evaluated using a separate testing file that will call your methods with various valid inputs.

- All source code must adhere to good Java programming style standards as discussed in lecture class and in the course textbook readings.
- All instructions and requirements must be followed; otherwise, points maybe deducted.

Examples

Output from FindTheSumsTester for a correctly implemented FindTheSums class:

```
Testing arrayToString method:
arrayToString(array1) test passed
arrayToString(array2) test passed

Testing horizontalSums method:
array1:
3 2 1 1
2 5 6 2
1 2 9 8
horizontalSums(array1, 7):
3 2 1 1
2 5 0 0
0 0 0 0
array2:
7 3 8 5 6 7 4 1 9 5
8 1 6 1 8 4 6 9 9 6
9 2 4 8 6 1 1 3 6 2
3 6 8 3 1 9 2 7 9 6
5 7 7 6 3 5 6 4 2 1
6 4 5 5 7 6 8 1 9 7
8 4 5 4 3 7 1 2 1 8
6 8 6 7 8 6 2 4 6 2
7 8 6 8 3 8 2 2 8 5
8 7 7 6 6 2 9 9 5 8
horizontalSums(array2, 20):
0 0 0 0 0 0 0 0 0 0
0 1 6 1 8 4 0 0 0 0
0 2 4 8 6 1 1 0 0 0
3 6 8 3 0 0 0 0 0 0
0 7 7 6 3 5 6 4 2 0
6 4 5 5 0 0 0 0 0 0
0 0 5 4 3 7 1 0 0 0
6 8 6 0 8 6 2 4 6 2
0 0 0 0 0 8 2 2 8 0
0 7 7 6 0 2 9 9 0 0
horizontalSums(array2, 25):
```

Lab 12 – FindTheSums

```
0 0 0 0 0 0 0 0 0 0
0 0 6 1 8 4 6 0 0 0
0 2 4 8 6 1 1 3 6 0
0 0 0 0 0 0 0 0 0 0
5 7 7 6 0 0 0 0 0 0
0 0 0 0 0 0 8 1 9 7
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 8 6 8 3 8 2 2 8 5
0 0 0 0 0 2 9 9 5 0

Testing verticalSums method:
array1:
3 2 1 1
2 5 6 2
1 2 9 8
verticalSums(array1, 22):
0 0 0 0
0 0 0 0
0 0 0 0
array2:
7 3 8 5 6 7 4 1 9 5
8 1 6 1 8 4 6 9 9 6
9 2 4 8 6 1 1 3 6 2
3 6 8 3 1 9 2 7 9 6
5 7 7 6 3 5 6 4 2 1
6 4 5 5 7 6 8 1 9 7
8 4 5 4 3 7 1 2 1 8
6 8 6 7 8 6 2 4 6 2
7 8 6 8 3 8 2 2 8 5
8 7 7 6 6 2 9 9 5 8
verticalSums(array2, 14):
0 0 8 5 6 0 0 0 0 0
0 0 6 1 8 4 0 0 0 6
0 0 0 8 6 1 0 3 0 2
3 0 0 3 1 9 0 7 0 6
5 0 0 6 3 5 6 4 0 1
6 0 0 5 7 0 8 1 0 7
8 0 0 0 3 0 1 2 0 0
6 0 0 0 8 6 2 0 6 0
0 0 0 8 3 8 2 0 8 0
0 0 0 6 0 0 9 0 0 0
verticalSums(array2, 33):
0 0 8 0 0 0 0 1 9 0
0 0 6 0 0 0 0 9 9 0
0 0 4 8 0 0 0 3 6 0
0 0 8 3 0 9 0 7 9 0
```

Lab 12 – FindTheSums

0 0 7 6 0 5 0 4 2 0
0 0 0 5 0 6 0 1 9 0
0 0 0 4 0 7 0 2 1 0
0 0 0 7 0 6 0 4 6 0
0 0 0 8 0 0 0 2 0 0
0 0 0 0 0 0 0 0 0 0