Lab 9 - 2D Arrays

1) Navigate to the **Labs** package in IntelliJ and create a package named **Lab9**

2) Drag the **Lab9** file provided into **Lab9** package**.** In this file you will see two methods that are blank. The first method is **addTo10** (below).

```
public static void addTo10(int[][] array){
 //Your code here
}
```

This method accepts a 2-dimensional array of integers. The array that is passed in can be any size. However, each row within the 2-dimensional array will have exactly one element with a value of 0. For each row, you will modify each 0 element so that the sum of all numbers in each row is 10. No other elements should be modified. A 2-D array for testing the addTo10 method is provided in the main method. The 2D array is named addTo10Input. Here is an example:

3	5	-8	19	0	17

turns into

3

since 3 + 5 – 8 + 19 – 26 + 17 = 10

Your method will print the array with the modified element, with each row on one line and each element separated by a space. There is also one matrix in the main method that will be used to test your method, matrix.

3) You will see another method in **Lab9** called **setHints** (below).

```
public static void setHints(int[][] array){
      //Your code here
}
```

4) Build this method so that it accepts a 2-dimensional array of integers containing -1s or 0s in each element, -1 indicates the presence of a mine and 0 is an open space.

You will modify this array according to the following (minesweeper) criteria:

- Walk through the array row by row.
- If a -1 is found, this indicates that this square in the array contains a mine. The code must then increment each surrounding square by one to indicate the presence of this mine unless that surrounding square has a value of -1.
- If the surrounding square is a -1, then that square is not modified.
- Continue this for all mines, or values of -1, present in the array.

Write a class “Program1” that has the method “hasDuplicates” (below) that accepts an array of strings

and returns a boolean indicating whether or not the given array contains any duplicates (case insensitive). The array should not be modified.

public static boolean hasDuplicates(String[] array)

You have two methods that can help you with this: the isInBounds() and isBomb() methods

Program 2

below this method. The isInBounds() method will check to see if the surrounding positions of a mine are in the bounds of the array. It will prevent you from going outside the index of

Write a class "Program2" that has the method "addTo10" (below) that accepts a 2 dimensional array of

the array. The isBomb() method will check to see if the surrounding square is a bomb, and

integers. Each row will have exactly one element with a value of 0. For each row, modify this element so

therefore, will not be changed.

that the sum of all numbers in the row is 10. No other elements should be modified.

public static void addTo10(int[][] array)

Your method will print the array with the modified element, with each row on one line and each element separated by a space. There are also two matrices in the main method that will

Program 3

be used to test your method, matrix2 and matrix3.

Write a class "Program3" that has the method "setHints" (below) that accepts a 2 dimensional array of integers containing -1s and 0s and modifies it according to the following (minesweeper) criteria: to hint

The figure below illustrates the state of an array at the beginning of the method (left) and at

of its presence, for each mine (a value -1) increment by 1 the value of all its surrounding squares; if

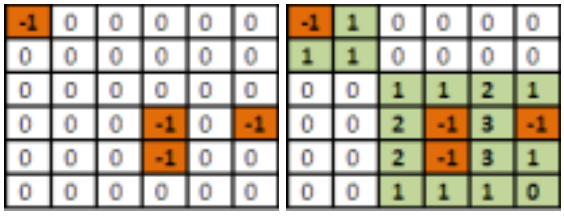
the end of the method (right) after all the hint values have been added. Here is a hint: for

another mine exists in one of these squares, its value is left unchanged.

each element around each mine, you need to check that it is inbounds and that it is not a mine. If neither, then increment the value by one. Have a method that checks inbounds and

public static void setHints(int[][] array)

not a mine and call those methods for each element surrounding the mine when found. The figure below illustrates the state of an array at the beginning of the method (left) and at the end of the method (right) after all the hint values have been added.



JUnit Tests

Each program above has a JUnit test class that can be downloaded from Scholar. These JUnit classes are

5) You can run your methods by running your main method that will run these methods with the complete and should not be modified. In addition, your programs should not hard-code any

answers.arrays provided in main. You can add output lines to output the results.

6) Fully test your methods by running `Lab9Test.java`. Correct your `Lab9.java` file so that each test runs successfully.

7)

8) Submit your `Lab9.java` file to Gradescope and ensure that all tests pass.

9) Take a screenshot of the file `Lab9Test.java` and upload the screenshot along with the `Lab9.java` file to the submission area in Canvas.