CSCI 2251 – Programming Assignment
Matrix Addition – Part 2 of 2

Instructions for Part 2

Put it all together to read in the data from file, spawn four threads, and allocate the task of summing each quadrant pair to a separate thread.

Consider which methods should be relocated from `Main` to `ThreadOperation`. What tools does `ThreadOperation` need to have access to in order to do its job?

After the threads complete their computation, the results need to be stored in the matrix C, another 2-dimensional array variable in main.

Your program should work for any size matrices.

In `ThreadOperation` write a method named `getQuadrantIndexes` that determines the indexes needed to iterate over one of the four quadrants. For instance, your method might take as input the row count, column count, and a quadrant String, and then return 4 numbers in an array: row start, row end, column start, column end. Although I'm demonstrating this method using a String to indicate the quadrant, an integer would also work fine, and an enum with four values would be best.

```
public static int[] getQuadrantIndexes(int rows, int columns, String
quadrant)
```

Called as `int[] indexes = getQuadrantIndexes(rows, columns, “upper left”);`

There are many different (and some better) ways to get the indexes, but this is the way that I think will make sense to the most people.

So how do you actually calculate the indexes needed? You will need four conditions (if, elseif, elseif, else) for the four quadrants. Figure out the pattern based on the following examples:

Example 1

	6 columns	
8 rows	Row indexes from 0 to 3 Column indexes from 0 to 2	Row indexes from 0 to 3 Column indexes from 3 to 5
	Row indexes from 4 to 7 Column indexes from 0 to 2	Row indexes from 4 to 7 Column indexes from 3 to 5

Example 2

	12 columns	
7 rows	Row indexes from 0 to 2 Column indexes from 0 to 5	Row indexes from 0 to 2 Column indexes from 6 to 11

	Row indexes from 3 to 6 Column indexes from 0 to 5	Row indexes from 3 to 6 Column indexes from 6 to 11

Example 3

	5 columns	
9 rows	Row indexes from 0 to 3 Column indexes from 0 to 1	Row indexes from 0 to 3 Column indexes from 2 to 4
	Row indexes from 4 to 8 Column indexes from 0 to 1	Row indexes from 4 to 8 Column indexes from 2 to 4

Example 4 Fill-in-the-blanks

	C columns	
R rows	Row indexes from 0 to _____ Column indexes from 0 to _____	Row indexes from 0 to _____ Column indexes from _____ to _____
	Row indexes from _____ to _____ Column indexes from 0 to _____	Row indexes from _____ to _____ Column indexes from _____ to _____

`Main.java` should be organized as follows (Strongly consider using the following notes as comments):

- Your `main` method opens a text file using the file name from the command line, and reads in the number of rows, the number of columns, and two matrices, A and B, into two 2-dimensional array variables.
- Instantiate four `ThreadOperation` objects and pass them the information they need to sum up paired quadrants, including a reference to a result matrix C. Note that C should have the same dimensions as A and B.
- Start up all the threads and use join to make sure they finish before printing.
- Print out the summed matrix.

ThreadOperation.java Organization

I recommend formatting the ThreadOperation constructor as follows:
<<constructor>>ThreadOperation(A : int[][], B : int[][], C : int[][], String quadrant)
run() : void

A, B, and C all refer to complete matrices (no sub-matrices) of the same size. As long as the Thread is not accessing the same row and column as another Thread, there's no problem!

UML Diagram for Matrix Addition Part 2

You tell me!

For part 2 you must turn in a UML diagram of your code, including the ways you modified `Main` and `ThreadOperation` to complete the assignment.

Compilation and Execution

I will test your program as follows:

```
  javac *.java
  java Main matrix1.txt
or
  java Main matrix2.txt
or
  java Main matrix3.txt
```