Overview
This lab continues the processing for a game of Life that was started in the Life Matrix assignment. We have a matrix of organisms in a rectangular area, represented by a matrix of booleans. The organisms reproduce and die out according to rules based on population density. At each time advance some new organisms may be born and existing ones may die. Existing organisms may die due to overcrowding or isolation. New ones are born when nearby cells reflect an appropriate population density.

The game is thus parameterized by the particular population density ranges that allow organisms to be born or to continue to thrive. Each cell has up to eight neighbors (above, below, left, right, and four diagonals). Thus, if you count the cell as well, there are nine cells in its neighborhood. For a birth to occur, the cell must be empty and the neighborhood density be in the birth range. The birth range can be given as two integers from zero to nine, birthLow and birthHigh, so that a birth will occur if the cell is empty and the neighborhood population is at least birthLow and no more than birthHigh. Similarly, a death occurs in a cell if there is an organism there and the neighborhood has less than the minimum population for survival, or greater than the maximum. Hence there is a live range provided as two integers from zero to nine, liveLow and liveHigh.

The border of the area is not compatible with life, so the top and bottom rows and the left and right columns will never have organisms.

This part of the assignment will take the matrix built in the previous assignment and make discrete time increments to advance the state of the game.

Specifications
Your program will read the parameters from standard input. Do NOT include any prompts – prompts will cause the program to fail when tested against my output!

This second phase begins as the Life Matrix program began by reading the number of rows, the number of columns, and a seed to randomly fill the matrix. The number of rows and columns are integers, the seed is a long. The new inputs for this part are the values for the ranges, birthLow, birthHigh, liveLow, and liveHigh. You must read the values in exactly that order with no other input. After building the matrix as in the previous assignment, you should print the matrix, as in the previous assignment, and then repeat for five iterations of updating the matrix and printing the result.
To calculate whether an entity lives at a given cell for the next iteration, you will need to calculate the number of entities in the neighborhood and compare the count with the birth range (if the cell is empty) or live range (if there was already an entity there). Note that the count must be based on the values as of the previous iteration. This means that you will need to make a copy of the matrix before doing the updates so you can count entities in neighborhoods based on original values.

You should run five update and print cycles.

Instructions
You should use several public static methods with appropriate parameters in this assignment. Choose methods that are clean and do not mix functionality. That is, you should not print from inside a method that will update the state, nor should you change the state in a print method. You should not use any non-local (class or instance) variables.

To count entities in the neighborhood, you will need to make a copy (clone) of the matrix. This is tricky because asking the matrix for its clone, with myMatrix.clone(), will result in a matrix with the same rows so that updates to the original rows will change the rows in the clones! What you need to do is create a new matrix by cloning the original and then cloning the rows of the original as in:

```java
boolean[][] myNewMatrix = (boolean[][])myMatrix.clone();
for (int row=0; row < myMatrix.length; row++) {
    myNewMatrix[row] = (boolean[])myMatrix[row].clone();
}
```

Remember that the borders of the area are toxic so entities cannot be born along the border. So in doing the update, start at row 1, column 1 and stop one less than the last row and column of the matrix.

Check your work. Have you tested your program with large and small matrices? Have you included your name in the comments and expanded the comments to include a description of the problem? Did you use methods? Are their parameters clear, appropriate, and minimal? Does each method have a short comment describing what it does and perhaps pre- and post-conditions? Does the program's indenting match its control structure? Do the for loops have appropriate bounds?

Submission
Submit this as the update lab using autolab.

Sample Input
6 8
7
3 8
3 8

Sample Output