Introduction
The game of Life simulates organisms that inhabit an area. The organisms reproduce and die out according to rules based on population density. The area is modeled in a 2-dimensional matrix with each entry in the matrix representing a cell in the living area. Time advances in discrete time units. 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. You can see one simulation of this game here: http://www.bitstorm.org/gameoflife/

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 population 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 simply initialize and print the values for a start state in the game of Life. You do NOT have to implement the full game in this phase.

Specifications
Your program will read the parameters from standard input. This part of the program will have three inputs: the number of rows, the number of columns, and a random-number-generator seed to randomly fill the matrix. The number of rows and columns are integers, the seed is a long. You must read the values in exactly that order with no other input. There should be NO prompt messages printed by your code.
You are to construct a two-dimensional matrix of booleans of the given size. You should fill the matrix with false. Then create a Random object initialized with the provided seed and fill the interior of the matrix row by row from left to right within each row. **Do not place a new value in the first or last row, or the first or last column of the matrix as they may not contain an entity (and hence should remain false).** You should use the nextBoolean method of the Random object to get each value. This will provide a matrix where each value (other than on the edge) is equally likely to be true or false.

Finally, print the matrix by printing a - for each false and a @ for each true. Include the outside rows and columns in the printing. Print a space before and after each – or @.

**Instructions**

You must name your file and class "Life" for grading as well as compatibility with later assignments (so the full file name will be Life.java). You should use several public static methods with appropriate parameters in this assignment. Remember that each function should do one thing and do one thing well. You should not use any non-local (class or instance) variables.

*Check your work thoroughly.* 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? Does each method have a short comment describing what it does and perhaps pre- and post-conditions? Did you pass the parameters appropriately? Does the program's indenting match its control structure? Do the for loops have appropriate bounds?

**Sample Input**
6 8
7

**Sample Output**
- - - - - - - - -
- @ @ @ - - - -
- @ @ @ @ - @ -
- - - @ @ - @ -
- @ - @ - @ @ -
- - - - - - - - -

**Grading**

Note that 25 of the 50 points will be graded automatically when you submit your program. This portion of the grade for this lab will be "all or nothing". Be very careful about your formatting – the grader is matching your output
against a solution file. If your program is being marked as incorrect, check the following:

- You should not have a leading space (the first character of each line should be a ‘-’).
- You SHOULD have a trailing space (the last character of each line should be a space).
- You should have a trailing newline character (so it will look like the last line of your output is blank).
- You should have a space after each output character (so that there is a column of spaces between each column of characters…how many?).

The remaining 25 points will be individually graded based on design (choice of methods, parameters, return values), format and documentation.

**Submission**
Submit this as the **matrix** lab using the GRASST submission system (in the lab).