Life Object (lifeObject)
CS 162 - Spring 2018
Due: 11PM Monday, February 12
100 points

Overview
This lab extends the game of Life assignment that was started in the Life Matrix assignment and continued in
the Life Update assignment. You will convert your Life class to allow for the creation of Life objects. These
objects are used by interfaces that are available in ~katz162/labs/life. The interfaces provide console and
graphical views. You should build the methods so that they make use of the already existing functions that
you wrote for earlier parts of the Life assignment.

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 class will provide a constructor of seven parameters, a seed for the random number generator, the
number of rows and columns in the area, the lower and upper bounds of the birth range, and the lower and
upper bounds of the live range. The seed is of type long, the other parameters are int. It should initialize an
object that has a matrix filled with random boolean values and retains the ranges for update processing.

The class should provide a method, update, that advances the state by one time unit. This method has no
parameters.

The class should provide a method, world, that returns a boolean matrix that describes the current state. This
method also has no parameters.

The class should still operate as in the last assignment when started from the main method of the Life class.

Instructions
You should not need to modify the static methods that you wrote for earlier parts of the assignment. You
should call them with appropriate parameters. However, if you did not build appropriate methods that do one
thing well, you may have to redo your code to include them. Your original program should still work if you
run the program starting with the main in the Life class.

Copy the files in ~katz162/labs/life to your project source code using:

    cp ~katz162/labs/life/* .

from inside your source code directory (where you submit from, and where Life.java is). Note that the space
and dot following the * are required. Then open eclipse, select the Life project, and use the File/refresh menu
cmdmand. Eclipse should update the project information to match the actual file system and thus include the
new files.

At this point, the interface files will not compile cleanly. That is because it expects to be able to use a Life
object which you have not yet built. DO NOT change the interfaces. They should work after you modify the
Life class.

Your Life class must have constructors, accessors, and modifiers that precisely match the requirements of the
interfaces. You may not modify the interfaces. Only the Life.java file will be submitted. Therefore, your class
must precisely interact with these interfaces so my testing will work.

You will need to declare instance variables to hold the matrix and the two ranges (four ints). Your instance
methods can then pass them as needed to the static class methods you have previously written.

You can then write the constructor, update, and world methods as described in the Specifications section.

The constructor has parameters that should be checked for validity. Row and column should be at least one.
The range values should be between one and nine inclusive. A high value that is less than a low value is legal
and specifies an empty range. If a value is not valid, you should throw an IllegalArgumentException with a
string that states what the value is and why it is illegal. You can do this with code such as:

    throw new IllegalArgumentException("Row must be positive, not "+ row);

The world method should return a copy of the state matrix rather than just return the instance variable so
that users of the class cannot modify the world. This is a speed versus strict protection of the representation
issue. Java does not provide a mechanism to provide fast access with strict protection.
Check your work. Have you tested your program with both interfaces? 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 lifeObject lab to my katz162 grader account.

Sample Input (console interface)
6 8
7
3 8

Sample Output (console interface)
- - - - - - -
- # # # - - -
- # # # # - # -
- - - # # - # -
- # - # - # # -
- - - - - - -
- - - - - - -
- # # # # - -
- # - - # # # -
- # # - - # # -
- # # # # # -
- - - - - - -
- - - - - - -
- - - - - - -
- # # # # # # -
- # # # # # # -
- # # # # # # -
- - - - - - -
- - - - - - -
- # # # # # # -
- # - - - - # -
- # - - - - # -
- # # # # # # -
- - - - - - -
- - - - - - -
- - - - - - -
- # # # # # # -
- # # # # # # -
- # # # # # # -
- - - - - - -
- - - - - - -