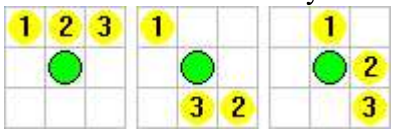
Programming Assignment: Life, Next Generation

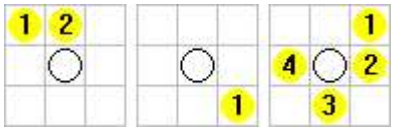
Overview

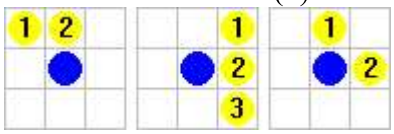
Life is a mathematical “game” invented by mathematician, John Conway. It became widely popular after it was published in a column in *Scientific American* in 1970. It is one of the most commonly programmed games on the computer.

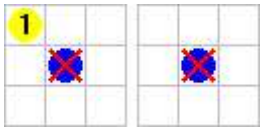
Life is an example of a *cellular automaton*, a system in which rules are applied to cells and their neighbors in a regular grid. Life is played on a rectangular grid of square cells. Each of these cells are either dead or alive. Here are the rules for Life:

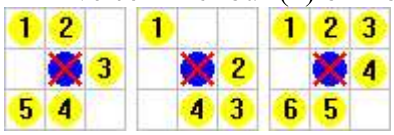
- Each cell has eight (8) neighbors.


- Once started, the state of a cell depends on its current state and the state of its neighbors.
- If a dead cell has exactly three (3) live neighbors, it becomes alive. This is a birth.


- If a dead cell does not have exactly three (3) live neighbors, it remains dead.


- If a live cell has two (2) or three (3) live neighbors, it remains alive. This is survival.


- If a live cell has zero (0) or one (1) live neighbors, it dies. This is loneliness.


- If a live cell has four (4) or more live neighbors, it dies. This is overcrowding.


These rules are applied to all of the cells at one time. That is, the new state of each cell is determined before changing the state of any of the cells.

Click here to see some additional notes on Life.

Objectives

- Manipulate data within an array.
- Print out the contents of a two-dimensional array.

Note: This part of the larger project is about data manipulation. The output using System.out.print(ln) and the lack of input are simplifications for this programming assignment. Later PAs will enhance this project, adding output (graphics) and providing a means to input a starting state.

Minimal Level

Download the starting point code:

- GameOfLife.java
- TestGameOfLife.java

This code will compile as downloaded, but it will not run. That's your task.

Life is designed to run in a grid of infinite size. John Conway initially created Life on a Go board, 19 x 19. We will use this simplification for our implementation.

For the Minimal Level, there are three tasks

- Implement GameOfLife
- Update TestGameOfLife
- Answer questions 1, 2, and 3

Implement GameOfLife

Create a class named **MyGameOfLife**. This class implements the GameOfLife you downloaded. It should have a single parameter-less constructor.

It will be easiest to create a "two-dimension" array as a private field (instance variable) in your class. For this project, the board size is fixed at 19x19, as noted above. The homework guidelines (and most style guides) state the non-static reference fields (instance variables) shall be initialized in the constructor. The getCellState and setCellState methods should be straightforward to implement.

For the starting point level, the nextGeneration method can simple be a *stub*. (A stub is a non-fulnctional method implementation that can be used to "fill in" for work to be completed later.) Here is the stub for nextGeneration:

```
public void nextGeneration() {
}
```

The largest single task for the Starting Point level is implementing the toString method of GameOfLife. This method returns a String representation of the 19x19 Game of Life board. Generate the string representation using a zeriod (.) representing a dead cell and a capital o (O) representing a live cell. Within the row, separate the cell values (. or O) with a space-bar space. Separate the rows with a new line character ('\n'). It is acceptable to use string concatenation for this method. The use of StringBuilder or StringBuffer is not required.

Update TestGameOdLife

The downloaded code for TestGameOfLife has several issues:

- It contains relatively few descriptive comments
- It does not run.

Add comments
Add descriptive comments to the source code. This shall include comments for the class as well as each of the methods in the class. If there were fields in this class, they also shall have comments.
This aspect of the PA is reflected in the Documentation portion of the score.

Comment for the class
The comment for the class shall appear above the class declaration. It is laudable if this is a documentation (JavaDoc) comment. The class comment shall contain a brief description of the overall purpose of the class, one or two sentences. It shall contain your name, in the @author tag as appropriate. It shall also contain the level at which you want this PA graded, in the @version tag, as appropriate. (See the homework guidelines for details.)

Comments for the methods
Each method in the class shall have a comment. The comment for the method shall appear above the method declaration. It is laudable that this is a documentation (JavaDoc) comment. The method comment shall conatin a brief description of the overall function of method, a sentence or two. It shall contain descriptions of any parameters and / or return of the method, in approapriate tags, as appropriate.

Fix the application so that it runs
Remove the comment from the line:

```
life = MyGameOfLife();
```

The class will now use your new implementation of the GameOfLife interface.

Expected results
If you have properly implemented things, the TestGameOfLife application should generate the following print out.

```
Starting point:
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . O . . . . . . . . O O . . . . . .
. . . O . . . . . . . . O O . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . O O . . . . . . . . . . . .
. . . . . O . . O . . . . . . . . . . .
. . . . . O O . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . O O O . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . O . . . . . . . . . . . . . .
. . . O . O . . . . . . . . . . . . . .
. . . . O O . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
```

Notice that there is a blank line at the end of the output. This board state is set by the setBoard method. It contains four named patterns: a block, a beehive, a blinker, and a glider. The fifth pattern has two live cells arranged vertically. It doesn't have a name.

Questions 1, 2, and 3

These three questions deal with the initial lines of the application method. The answers to the questions shall appear in the record file.

1. What is the meaning of the keyword **null**?
2. What is the error (exception) that occurs when the TestGameOfLife code as downloaded is run?
3. The TestGameOfLife code as downloaded requires the initialization of the local variable life to mull. It would not compile if the initial value were omitted from the variable declaration. Why?

In the report file, number the answers to these questions. It is not necessary to repeat the question.

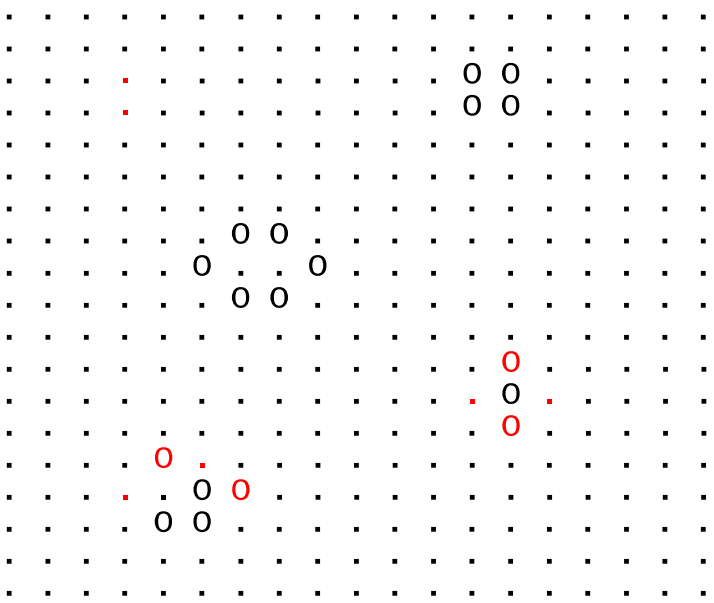
Standard Level

For the Standard Level, in addition the work for the Minimal Level, complete three tasks.

- Implement nextGeneration
- Update TestGameOfLife
- Answer question 4

Next Generation

Replace the stub for nextGeneration in MyGameOfLife. Given the starting point code, the second generation would appear as follows:



In this example, the changed cells are shown as red characters on a yellow background. Needless to say, there is no way to indicate which cells changed in your output. Just use System.out.print(ln) to output the updated matrix.

The nextGeneration method shall not instantiate any array objects. All array objects shall be instantiated in the constructor for MyGameOfLife. The state for any given cell cannot be updated until the next state for all of the cells in the board is determined. Here are two approaches to implement nextGeneration.

Approach 1:
Use two array objects, one to hold the current state of the boart and one as a work area. Use the values in the current array to determine the state for each cell in the next generation, storing those values in the work array. After all of the states for the next generation have been determined. Update the references. That is, store the reference to the current array in a temp variable. Update the current refernce to access the work array. Then

update the work reference to access the previous current array, now stored in the temp variable. This is much more time-efficient than copying all of the values from the work array into the current array.

Approach 2:
Use a single array object. Create additional "transitional" states for cells: BIRTH (dead bocoming alive) and LONELINESS and OVERCROWDING (alive becoming dead). As you determine the state for the next generation, mark the cells which change using one of these new transitional states. Then a BIRTH cell counts the same as a DEAD cell for determining the next generation. Similarly, a LONELINESS or OVERCROWDING cell counts as ALIVE. After all the next generation cells have determined, go through the array again and update the transitional states, so BIRTH becomes ALIVE, and LONELINESS and OVERCROWDING become DEAD. This is less time-efficient than approach 1, but it is more space-efficient. It also sets up for Challenge level work in a later PA for Game of Life.

Update TestGameOfLife

Update the Application Method:
Remove the remaining comments in the application method to enable the code to calculate and display the next generation.

JavaDoc comments:
For the Standard Level, the class and method comments shall be JavaDoc comments. For successful completion of this task. there shall be no error or warning messages from the JavaDoc compiler.

The following tags shall appear in the class JavaDoc comment: @author, with your name, and @version, with the grading level.

Question 4

4. The call the System.out.println that prints the board is different in the newly reinstated code. The argument is simply life, rather than life.toString. How does this work? (hint: Which overload of println is being called?)

Challenge Level

For the Challenge Level (*extra-credit*), in addition to the Standard Level, complete the following two tasks.

- More generations
- Answer Question 5

More Generations

Update the application method to display 7 generations of the Game of Life, the starting point and 6 new generations. Use a loop to control the calculation and display the new generations. Change the labels as follows:

> "Starting point:" becomes "Generation 0:"
> "First Generation" becomes "Generation 1:"

Follow the same pattern, "Generation 2:", "Generation 3:' … "Generation 6:"

Since you will be printing a largish (19x19) grid of values each time, use JOptionPane.displayMessageDialog to control the output, pausing between each generation. For the arguments, you can use (null, "Click OK to continue").

Notice that after 4 generations, the 5 live cells of the glider will return to its original configuration relative to each other if the next generation algorithm is correct.

Question 5

5. Temporarily increase the number of generations large enough so that the glider runs into the lower edge of the board. What happens to the glider after it encounters the edge? (This question is asking for the "fate" of the glider several generations "touching" the edge of the board.)

Remember to reset the total number of generations displayed to 7 before you submit the file.

Written Report

Write a report in which you describe:

- how did you go about starting this project?
- what works and what doesn't?
- the surprises or problems you encountered while implementing this application
- the most important thing(s) you learned from this assignment
- what you would do differently next time
- the answers to the questions as specified for the PA level

I expect a clear report that has some thought in it. It will be easiest if you take notes about the process as you work on the assignment, rather than waiting till the end to start the written report.

Grading Criteria

Functionality: (7 point, Standard level)
4/ Minimal level, implement GameOfLife to display initial configuration, comments
7/ Standard level minimal *plus* nextGeneration
9/ Challenge level, standard *plus* 7 generations with showMessageDialog

Style: (3 points)
conforms to the homework guildelines

Documentation (5 points):
* JavaDoc and other comments
* written report
* answers to specified questions depending on level
  - Minimal, 1 - 3
  - Standard, 1 - 4
  - Challenge, 1 - 5

Submission

- Two (2) Java source code files (.java)
  - MyGameOfLife.java (new file)
  - TestGameOfLife.java (updated)
- One (1) ASCII, plain-text report file (.txt)

Backk