Mini-project for lab working groups: GoL

Deliverables:

- Submit your source code to http://deei-mooshak.ualg.pt/~jvo/ (Problem H) and
- your report (within a zip file) as TP3

  to http://www.deei.fct.ualg.pt/POO/Entregas/

  including:

  i) The problem id, your group number, and its elements;

  ii) Your description of the problem and the approach followed to address it;

  iii) The analysis UML class diagram

  iv) The unit tests developed

  v) All design options taken

  vi) Javadoc

  vii) The implementation UML class diagram (see www.objectaid.com)

  viii) Any relevant conclusion or remark

  ix) Bibliographic references used, if any

Up to May 11, 2020

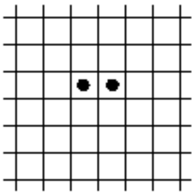
The problem

The Game of Life (GoL), introduced in 1970 by J. H. Conway, is a simulation of a borderless board, where each cell has a binary state of living or dead. Occupied cells are therefore addressed as *living* and free cells as *dead.*

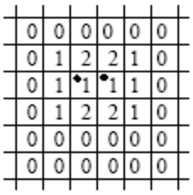
The board evolves at each generation according to the **transition rules**, in such a way that a cell can change its state according to the number of living neighbour cells.

Living neighbours

The neighbours of a cell are its 8 adjacent cells. The figure below shows in b) the counting of living neighbours for each cell, for a) a board with only 2 living cells represented by the black dots:



a)



b)

Transition rules

The next algorithm, computes the next generation of cells, given an initial cell distribution.

```
For each cell in the initial board,
      -  IF living cell and the number of living neighbours is 2
         or 3 then the cell lives in the new board; it dies
         otherwise
      -  IF dead cell and the number of living neighbours is 3
         then the cell lives in the new board; it remains dead
         otherwise
```

More information on GoL, as well as an online simulator, is available from:

https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life
https://bitstorm.org/gameoflife/

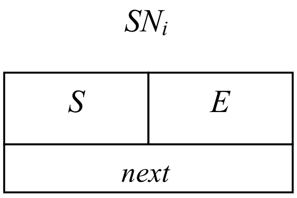
Task

Develop a Java program that given an initial board (cell distribution) shows the successive distributions of cells when the GoL transition rules are applied generation after generation.

REQUIREMENTS

- Employ the principles and techniques of object-oriented programming.
- Use test-driven development.
- Unexpected conditions that lead to invalid states should resort to exceptions
- The game board might be naively implemented as a bidimensional array. However this will waste unnecessary space for large boards, so you should implement the board as a sparse bidimensional array of cell that only saves living cells.

- In particular, the sparse array should be implemented using sentinel and data nodes. The sentinel nodes will form a circular linked list. The *i*-th sentinel node SN_i will have the following *ad hoc* schematic representation:



Where its members have the following meaning:

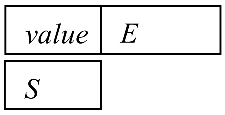
next - reference to the next sentinel node;

E – a reference to either the first data node of row *i* of the sparse array or to SN_i if row *i* does not have data nodes.

S – a reference to either the first data node of column *i* of the sparse array or to SN_i if column *i* does not have data nodes.

Therefore, the number of sentinel nodes equal the maximum of the number of rows and columns in the sparse array.

In turn, a data node will have, at least, the following information:



Where:

- *S* and *E* are references to either the next non-zero data node or to a sentinel node. The *E* of the last data node in a given row points back to the sentinel node of its row. Similarly, *S* of the last data node in a given column points back to the sentinel node of its column.

- value is of a *generic type* that in this particular problem holds a living cell.

NB: Any other approach to this problem, independently of its merit, will be marked with 0 (zero) values.

Evaluation of the unit tests

Multi-submissions to mooshak are allowed. However:

- Only the 3 first failed submission are free of charge; For the benefit of test driven development, you can submit as much accepted submissions as you like.
- After 3 failed submissions, the *i*-th failed submission has a penalty of 0,1*(*i-3)* over the final grade of this assignment; Example: a program was accepted at submission 6; i.e., it had 5 failed submissions with an associated penalty of 0+0+0+0,1+0,2 = 0,3v

Input

The first line of the input is a natural number *N* specifying the number of iterations or generations. After the first line, there will be as many input lines as the rows in the initial board. In each of these lines a 0 represents a dead cell and a 1 a living cell.

Output

The distribution of cells in every board of each generation up to generation *N*. One row for each board line; where a dead cell is presented by 0 while a living one is represented by a 1. Boards are separated by a blank line.

NB: A board can grow as needed, but should never be smaller than the initial board.

Sample Input 0: oscillator with borders

```
3
010
010
010
```

Sample Output 0

```
000
111
000

010
010
010

000
111
000
```

Sample Input 1: oscillator without borders

```
3
1
1
1
```

Sample Output 1

```
000
111
000

010
010
010

000
111
000
```

Sample Input 2: glider up left

```
4
111
100
010
```

Sample Output 2

```
010
110
101
000

110
101
100
000

0110
1100
0010
0000

1110
1000
0100
0000
```

Sample Input 3: vanishes

```
1
```

```
101
000
101
```

Sample Output 3

```
000
000
000
```

Sample Input 4: block

```
2
11
11
```

Sample Output 4

```
11
11

11
11
```

Sample Input 5: fade right without borders

```
3
110
001
110
```

Sample Output 5

```
010
001
010

000
011
000

000
000
000
```

Sample Input 6: fade left with borders

```
3
000000
001100
010000
001100
000000
```

Sample Output 6

```
000000
001000
010000
001000
000000

000000
000000
011000
000000
000000

000000
000000
000000
000000
000000
```