The Game of Life
Created by Michael Overton & Samuel Marateck

Description

Consider a two dimensional array with **M** rows and **N** columns. This represents the world in which some organisms live. Each of the **M** by **N** cells in this array is either occupied (if an organism lives there) or is vacant. No more than one organism can live in any one cell at any time. Each cell, except those on the boundaries (the edge of the world), has exactly eight neighboring cells (above, below, left, right, and four diagonals). The cells on the boundaries have less than eight neighboring cells.

Initially, there is a given population of organisms occupying certain of the cells. At each succeeding generation, the organisms reproduce and die as follows:

- Each organism of the current generation survives to the next generation if, and only if, it has 2 or 3 neighbors (a neighbor is an organism that lives in a neighboring cell). Otherwise, it dies and its cell becomes empty in the next generation. Note: it dies if it is “lonely” or “overcrowded”.

- Each vacant cell in the current generation becomes occupied by a new organism in the next generation if, and only if, it has exactly 3 neighbors. Otherwise, it remains vacant in the next generation.

For example, suppose the initial world (the “zero”th generation) is as follows, using X to indicate the occupied cells and blanks for the vacant cells:

```
          X X
XXX                 XX        XXXX
          X X       XX
```

Then the next generation is

```
 X                            XX
 X                  XX        XX
 X                  XX        XX
```

On the input file, the blanks are represented by dots.

Design and Implementation

Write a program to play this Game of Life. Your program should read the initial world from a file (see below) and repeatedly generate new generations, prompting the user each time to see if he or she wants to see the next generation or terminate the program. Also, the program should terminate automatically if the world becomes empty, displaying a message accordingly (this will happen for **life3.dat**).

Use two-dimensional arrays of type **char** to store the old and new generations respectively. To keep things simple, assume that **M=25** and **N=75**, i.e. the world has 25 rows and 75 columns, and define these in a **final** statement before declaring your array variables.

Data files for testing your program are provided. You should make sure your program works correctly on all these files, and you should also try your own test data. If you type **.dat** after the file name, it will be saved as **dat** file. Thus your program would read for instance, **file0.dat**. The program should prompt the user for the name of the input file (use a String variable).

Empty cells are represented with dots and occupied cells with X's. Read the data and assign the data to a two-dimensional array using two nested **for** loops. The outer loop reads a string consisting of the data on a given line of the input file. The inner loop assigns the data to a given row of the array. In order to read the file, you will have to use Scanner similarly to the way it is used in **FileReaderExample.java**.

Programming Details

You need two 2-dimensional arrays---the new generation should be created based on information from the old generation. If you use only one array, you will find that your old generation is overwritten by the new one before you have finished using all the information you need from it.

You should create a “border” of cells that always stay empty by declaring your arrays from 0 to **M+1** and from 0 to **N+1**. This way, the cells on the edge will also have eight neighboring cells and won't need special treatment.

Include at least two non-void methods. One should take a world and the coordinates of a cell and return the number of neighbors (organisms in neighboring cells) that the cell has. The other should take a generation array and return a Boolean value that tells whether or not the world represented by the array is empty.

Write all generations to the screen (don't forget the generation numbers), with a prompt to the user to type a key when ready to see the next one. Once your program is working, also terminate if the world is the same as the *previous one* (this will happen for **life4.dat**).

Grading Rubric

Points	Criteria
2	Read in .dat files correctly
4	Correct implementation of generation algorithm
3	Terminates appropriately for all end conditions
1	Good style demonstrated in the code; sensible formatting