**Description**

This is a console application in Java. The application is a game called **`GameOfLife`**, which runs continuously once you start the game (*infinite looping program*). It has been used as the screen saver in legacy computers. More information can be found in https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life.

The game has a **`M`**×**`N`** board, where **`M`** is the number of rows and **`N`** is the number of columns. Each cell on the board can be a live cell, denoted by the symbol **`'#'`**, or a dead cell, denoted by a space **`' '`**. The initial pattern of the board is saved in the input file (**`Input.txt`**). The format of the file is as follows:

```
M
N
<R1> <C1>
<R2> <C2>
…
…
<RZ> <CZ>
```

where $\mathbf{R_i}$ and $\mathbf{C_i}$ refer to the row number and column number of the *i*-th live cell respectively. There may be one or more live cells recorded in the file. The following shows a sample input file and the corresponding board graphically.

| Input.txt | Board |
|---|---|
| 10<br>10<br>0 2<br>1 3<br>2 1<br>2 2<br>2 3 | |

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | # | | | | | | | |
| 1 | | | | # | | | | | | |
| 2 | | # | # | # | | | | | | |
| 3 | | | | | | | | | | |
| 4 | | | | | | | | | | |
| 5 | | | | | | | | | | |
| 6 | | | | | | | | | | |
| 7 | | | | | | | | | | |
| 8 | | | | | | | | | | |
| 9 | | | | | | | | | | |

The game runs as follows:

- The game firstly reads the input file and load the initial pattern.
- Every one second, the pattern will be updated using the following rules:
    1. Any live cell with two or three live neighbours survives in the next generation;
    2. Any dead cell with three live neighbours becomes a live cell in the next generation;
    3. All other live cells die in the next generation. All other dead cells stay dead in the next generation;
- Note: When counting the number of neighbours for a cell, which is at the edge, it treats the cells at the opposite edge as neighbours. For example, the neighbours of the cell at **(0,0)** include the cells at **(9,9)**, **(9,0)**, **(9,1)**, **(0,9)**, **(0,1)**, **(1,9)**, **(1,0)** and **(1,1)** if it is a **10×10** board.

Referring to the same example above, the following shows the number of neighbours of each cell:

**Input.txt**

```
10
10
0 2
1 3
2 1
2 2
2 3
```

**Board**

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | # | | | | | | | |
| 1 | | | | # | | | | | | |
| 2 | | # | # | # | | | | | | |
| 3 | | | | | | | | | | |
| 4 | | | | | | | | | | |
| 5 | | | | | | | | | | |
| 6 | | | | | | | | | | |
| 7 | | | | | | | | | | |
| 8 | | | | | | | | | | |
| 9 | | | | | | | | | | |

**Number of Neighbours**

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 2 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 3 | 5 | 3 | 2 | 0 | 0 | 0 | 0 | 0 |
| 2 | 1 | 1 | 3 | 2 | 2 | 0 | 0 | 0 | 0 | 0 |
| 3 | 1 | 2 | 3 | 2 | 1 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |

**Additional Requirement and Assumption:**

- **Zero marks will be given if the program is not able to compile.**
- You can decide to use or not to use the class **`Board.java`**. Not using **`Board.java`** will not result in any penalty or bonus.
- You may assume that input file must be valid.
- The following code will save the second of the current time to the variable s. For example, if the current time is 17:34:28. The value of s is 28.

```
LocalTime now = LocalTime.now();
int s = now.getSecond();
```

- The flow of the program is as follows:
  1. Read the first two lines of the input file **`input.txt`**
  2. Create the board and set all cells to dead
  3. Create a 2-D array to store the number of neighbours
  4. For each of the remaining lines of the input file:
     - Change the corresponding cell to live
  5. Print the board ( **`board.print()`** )
  6. while(true): *// Yes! It is an infinite-loop*
     - Every one second, the pattern will be updated:
       - Calculate the number of neighbors for each cell and update the 2-D array in Step 3.

- Update the live/dead cells for each cell in the next generation (**`board.getCells()[i][j]`**)
- Print the board

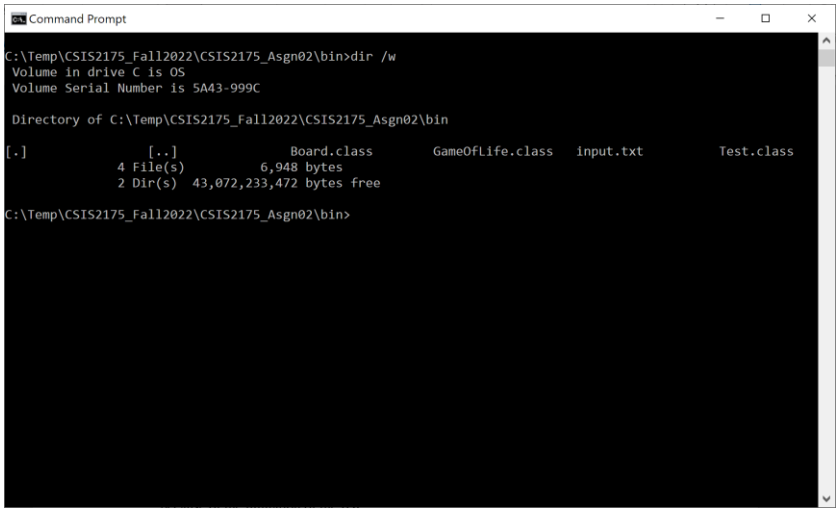
**Sample Run**

A video showing the sample run of the program using the above example can be found in Blackboard.

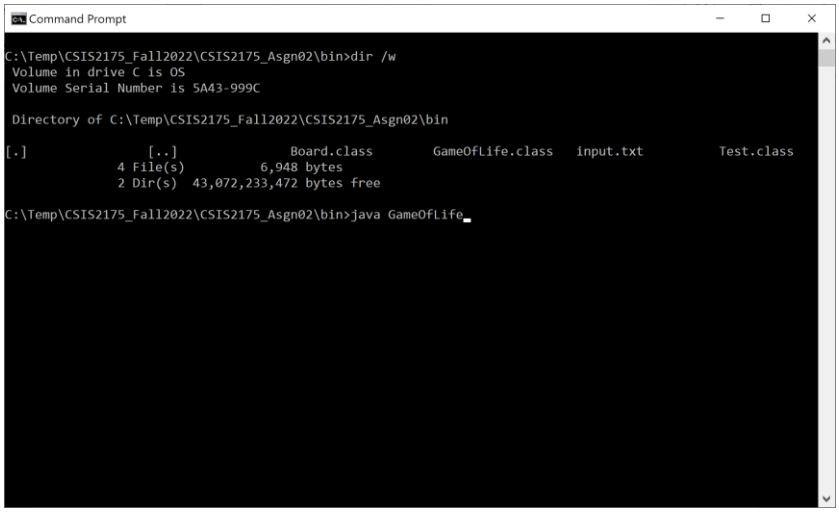
More pattens can be found in https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life.

**Notes on Testing the Program**

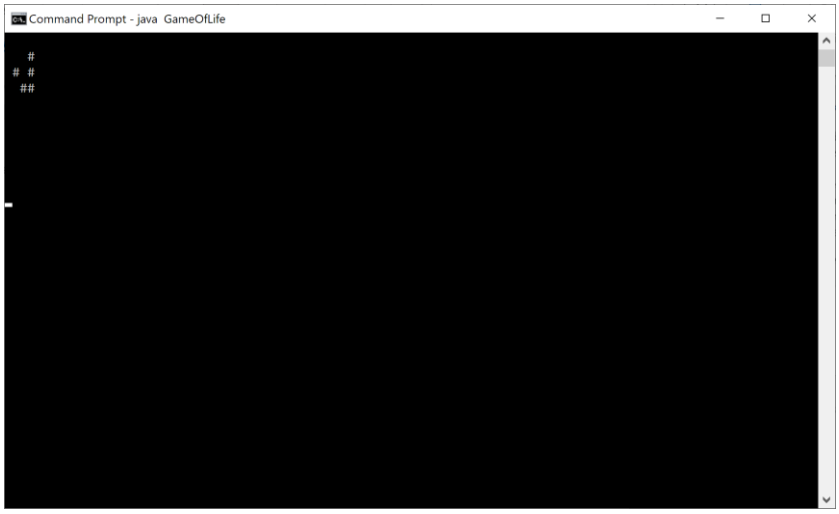
- If you test your program in Eclipse, the program cannot clear the console window because of the limitation of the IDE.
- If you have the **`input.txt`** and all **`.class`** files in the sample, you can test the program in Windows Command Prompt as follows. It can clear the console for every second.
    1. The folder contains input.txt and all .class files:



2. Run java GameOfLife in the Command Prompt:




3. The game should run properly:




4. Press Ctrl+C to stop: