```
public final class GameOfLife {

    // The value representing a dead cell
    public final static int DEAD    = 0x00;

    // The value representing a live cell
    public final static int LIVE    = 0x01;


    public final static void main(String[] args) {

        // test the game of life implementation
        GameOfLife gof = new GameOfLife();
        gof.test(4);
    }


    /**
     * Test the gameoflife implementation, change the array
     * values to test each condition in the game of life.
     *
     * the number of times the board should be played
     */
     private void test(int Iterations) {

        // the starting playing board with life and dead cells
         int[][] grid = {{DEAD, DEAD, DEAD, DEAD, DEAD,DEAD, DEAD, DEAD, DEAD,
DEAD},
                         {DEAD, DEAD, DEAD, LIVE, DEAD,DEAD, DEAD, DEAD, LIVE,
DEAD},
                         {DEAD, DEAD, LIVE, LIVE, DEAD,DEAD, DEAD, LIVE, LIVE,
DEAD},
                         {DEAD, DEAD, DEAD, LIVE, DEAD,DEAD, DEAD, DEAD, LIVE,
DEAD},
                         {DEAD, DEAD, DEAD, DEAD, DEAD,DEAD, DEAD, DEAD, DEAD,
DEAD},
                         {DEAD, DEAD, DEAD, LIVE, DEAD,DEAD, DEAD, DEAD, LIVE,
DEAD},
                         {DEAD, DEAD, DEAD, LIVE, DEAD,DEAD, DEAD, DEAD, LIVE,
DEAD},
                         {DEAD, DEAD, DEAD, LIVE, DEAD,DEAD, DEAD, DEAD, LIVE,
DEAD},
                         {DEAD, DEAD, DEAD, DEAD, DEAD,DEAD, DEAD, DEAD, DEAD,
DEAD},
                         {DEAD, DEAD, LIVE, LIVE, DEAD,DEAD, DEAD, LIVE, LIVE,
DEAD},
                        };

        System.out.println("Game Of Life");

        printGrid(grid);

        for (int i = 0 ; i < Iterations ; i++) {
            System.out.println();
            grid = getNextGrid(grid);
            printGrid(grid);

        }

        Scanner s=new Scanner(System.in);
         System.out.println("Enter Cell to check the state Cell");

         System.out.println("Enter Row");
         int R=s.nextInt();
```

```
         System.out.println("Enter Column");
         int C=s.nextInt();

         if(R<grid.length && C<grid[0].length)
         {
             int c=0;
             for(int i=0;i<grid.length;i++)
             {
                 for(int j=0;j<grid.length;j++)
                 {
                     if(grid[R][C]==0)
                         c=0;
                     else
                         c=1;

                 }
             }
             if(c==0)
                 System.out.println("===========Cell is Dead=========");

             else
                 System.out.println("===========Cell is Live=========");

         }
     }

     /**
      Print one grid to System.out

      grid The grid to be printed to System.out
      */
     private void printGrid(int[][] grid) {

         for (int i = 0, e = grid.length ; i < e ; i++) {

             for (int j = 0, f = grid[i].length ; j < f ; j++) {
                 System.out.print(Integer.toString(grid[i][j]) + " ");
             }
             System.out.println();
         }
     }

     /**
      * get the next game board, this will calculate if cells live on or die or
new
      * ones should be created by reproduction.
      *
         The current board field newly created game buffer
      */
     public int[][] getNextGrid(int[][] grid)
     {

         // The board does not have any values so return the newly created

         if (grid.length == 0 || grid[0].length == 0)
         {
             throw new IllegalArgumentException("Board must have a positive
amount of rows and/or columns");
         }

         int Rows = grid.length;
         int Cols = grid[0].length;

         // temporary board to store new values
```

```
        int[][] buf = new int[Rows][Cols];

        for (int r = 0 ; r < Rows ; r++)
        {

            for (int c = 0 ; c < Cols ; c++)
            {
                buf[r][c] = getNewCellState(grid[r][c], getLiveNeighbours(r, c,
grid));
            }
        }
        return buf;
    }

    // Get the number of the live neighbours given the cell position
       //the column position of the cell
       //the row position of the cell
       // the number of live neighbours given the position in the array

    private int getLiveNeighbours(int cellRow, int cellCol, int[][] grid) {

        int liveNeighbours = 0;
        int rowEnd = Math.min(grid.length , cellRow + 2);
        int colEnd = Math.min(grid[0].length, cellCol + 2);

        for (int row = Math.max(0, cellRow - 1) ; row < rowEnd ; row++) {

            for (int col = Math.max(0, cellCol - 1) ; col < colEnd ; col++) {

                // make sure to exclude the cell itself from calculation
                if ((row != cellRow || col != cellCol) && grid[row][col] ==
LIVE) {
                    liveNeighbours++;
                }
            }
        }
        return liveNeighbours;
    }



     //Get the new state of the cell given the current state and
     //the number of live neighbours of the cell.

     //The current state of the cell, either DEAD or ALIVE
     //The number of live neighbours of the given cell.

    private int getNewCellState(int curState, int liveNeighbours) {

        int newState = curState;

        switch (curState) {
        case LIVE:

            // Any live cell with fewer than two
            // live neighbours dies
            if (liveNeighbours < 2) {
                newState = DEAD;
            }

            // Any live cell with two or three live
            // neighbours lives on to the next generation.
            if (liveNeighbours == 2 || liveNeighbours == 3) {
                newState = LIVE;
```

```
            }

            // Any live cell with more than three live neighbours
            // dies, as if by overcrowding.
            if (liveNeighbours > 3) {
                newState = DEAD;
            }
            break;

        case DEAD:
            // Any dead cell with exactly three live neighbours becomes a
            // live cell, as if by reproduction.
            if (liveNeighbours == 3) {
                newState = LIVE;
            }
            break;

        default:
            throw new IllegalArgumentException("State of cell must be either
LIVE or DEAD");
        }
        return newState;
    }
}
```

==========

==========

Output:-

```
Game Of Life
0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 1 0
0 0 1 1 0 0 0 1 1 0
0 0 0 1 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 1 0
0 0 0 1 0 0 0 0 1 0
0 0 0 1 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0
0 0 1 1 0 0 0 1 1 0

0 0 0 0 0 0 0 0 0 0
0 0 1 1 0 0 0 1 1 0
0 0 1 1 1 0 0 1 1 1
0 0 1 1 0 0 0 1 1 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 1 1 1 0 0 1 1 1
0 0 0 0 0 0 0 0 0 0
0 0 1 1 0 0 0 1 1 0
0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0
0 0 1 0 1 0 0 1 0 1
0 1 0 0 1 0 1 0 0 1
0 0 1 0 1 0 0 1 0 1
0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 1 0
0 0 0 1 0 0 0 0 1 0
0 0 0 0 1 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 1 0 0 1 0
```

```
0 1 1 0 1 0 1 1 0 1
0 0 0 1 0 1 0 0 1 0
0 0 0 1 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0
0 0 0 1 1 0 0 0 1 1
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1 1 0
0 0 1 0 0 0 1 1 0 1
0 0 0 1 0 1 1 0 1 1
0 0 0 0 1 0 0 0 0 0
0 0 0 1 1 0 0 0 1 1
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
Enter Cell to check the state Cell
Enter Row
3
Enter Column
4
==========Cell is Dead=========
BUILD SUCCESSFUL (total time: 5 seconds)
```

Output - Employee (run)

```
Game Of Life
0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 1 0
0 0 1 1 0 0 0 1 1 0
0 0 0 1 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 1 0
0 0 0 1 0 0 0 0 1 0
0 0 0 1 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0
0 0 1 1 0 0 0 1 1 0

0 0 0 0 0 0 0 0 0 0
0 0 1 1 0 0 0 1 1 0
0 0 1 1 1 0 0 1 1 1
0 0 1 1 0 0 0 1 1 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 1 1 1 0 0 1 1 1
0 0 0 0 0 0 0 0 0 0
0 0 1 1 0 0 0 1 1 0
0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0
0 0 1 0 1 0 0 1 0 1
0 1 0 0 1 0 1 0 0 1
0 0 1 0 1 0 0 1 0 1
0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 1 0
0 0 0 1 0 0 0 0 1 0
0 0 0 0 1 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
```

Employee (run)

```
0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 1 0 0 1 0
0 1 1 0 1 0 1 1 0 1
0 0 0 1 0 1 0 0 1 0
0 0 0 1 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0
0 0 0 1 1 0 0 0 1 1
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1 1 0
0 0 1 0 0 0 1 1 0 1
0 0 0 1 0 1 1 0 1 1
0 0 0 0 1 0 0 0 0 0
0 0 0 1 1 0 0 0 1 1
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
Enter Cell to check the state Cell
Enter Row
7
Enter Column
5
==========Cell is Dead=========
BUILD SUCCESSFUL (total time: 1 minute 40 seconds)
```