INTRODUCTION TO COMPUTER SCIENCE II
PROJECT 2
The Game of Life, version 2

1 More abstracting, more capabilities

In Project-1, you implemented a basic version of the *Game of Life*; in Lab-4, you abstracted the `Cell` class to allow for different types of cells (*Conway* and *Highlife*).

Here, we're going to use more abstraction, adding capabilities and flexility that the original code didn't have. There will be more cell types, a choice of grid types, and a choice of user interfaces. Read on for details…

1.1 More cell types

Since you have already modified your code for multiple cell types, there are (at least) two more to add:

1. `ZombieCell`: This type of cell is always dead. The grid's `getCell()` method should be modified to **return one of these cells for out-of-bounds coordinates**. That method should no longer return `null`.
2. `MyCell`: Come up with your own rules. The rules are defined by the number of neighbors for a cell to be *born* (that is, the cell is currently dead, but becomes alive), as well as the number of neighbors for a cell to *survive* (that is, the cell is currently alive and stays that way). Conway cells, for example, are born with 3 live neighbors, and survive with 2 or 3 live neighbors; the shorthand *B3/S23* is commonly used. Highlife cells use *B36/S23*. Experiment with your own rule combinations to find one that does something interesting.

1.2 More grid types

Abstract the `Grid` class. Figure out which methods should become *abstract*—those methods that depend on the specific manner in which the `Cell` pointers are stored. Then create the following subclasses:

1. `Array2DGrid`: The old `Grid` class, but under a new, subclassed name and implementing only the *abstract* methods of its superclass. It is a grid implemented using arrays of arrays.
2. `Array1DGrid`: Implement a grid, but use a single, one-dimensional array (of pointers to `Cell` objects).

1.3 More user interface types

Abstract the `UserInterface` class. Again, figure out which methods should become *abstract*. Here, those methods are would do work specific to presenting the state of the grid in a particular manner should be abstract. Then, create the following subclasses that provide different user interfaces:

1. `DumbTextUserInterface`: The old `UserInterface` class, but under a new, subclassed name and implementing only the *abstract* methods of its superclass. It prints, as a log, the sequence of generations in rapid succession.

2. `SmartTextUserInterface`: Like the `DumbTextUserInterface`, but uses *Control Sequence Introducer (CSI) codes* to reprint the grid on top of itself. By printing CSI codes to the terminal in which the program is running, the cursor can be made to move in arbitrary directions, allowing the user interface to reset the cursor to the top of the grid output each time.

   For example, to make the cursor move up 5 lines, the following print statement would make that happen:
   ```
   System.out.print("\u001b[5A");
   ```
   Here, `\u001b[` is a special *escape sequence* that tells the terminal that special codes are to follow. (That sequence is the *CSI* defined on the above web page.) The `5A` is the code to direct the terminal to *move the cursor up* (`A`) by 5 lines (`5`).

3. `GraphicUserInterface`: A user interface that uses the *Swing* package (part of Java) to create a graphical window and draw the generation in it. More details to follow soon.

1.4 Changes to `getCell()`

The `Grid` method `getCell(i,j)` should be altered to behave as follows:

- If `(i, j)` is within the grid, return the `Cell` contained at that location.
- If `(i, j)` is in the *bounding frame*—the set of cells one position outside of the proper grid—then return a `ZombieCell` (described above).
- Otherwise, throw a `OffTheGridException` to indicate that `(i, j)` is outside of any range that should ever be requested.

2 Getting started

Create a new directory for your Project-2 work. Copy, from your Lab-4 directory, all of your `.java` and `.init` files. This previous work of yours is the starting point.

3 Your assignment

Write the classes described above. These new classes, as well as the new behavior of modified methods, may require changes elsewhere in the code. You are expected to identify those locations and make those changes. When done, a user should be able to run the program with their desired cell type, grid type, and user interface.

4 How to submit your work

Submit **all of your `.java` files**. Do so via one of the following tools:

- **Web-based:** Visit the CS submission system web page.
- **Command-line based:** On `remus/romulus`, use the `cssubmit` command at your shell prompt.

This assignment is due on Sunday, Mar-26, 11:59 pm.