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-3, 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 types and behaviors for cells and for grids. Read on for details...

1.1 Create more cell types

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

1. `ZombieCell`: This type of cell is always dead. If displayed, it should be shown as a `z`.
2. `MorphingCell`: If dead with 2 live neighbors, this cell becomes a `ConwayCell`; if dead with 3 live neighbors, it becomes a `HighlifeCell`. If alive with 4 or 5 live neighbors, this cell will remain alive. Otherwise, it will be dead. A live `MorphingCell` displays as a `#`; a dead one displays as `^`.

1.2 Create multiple grid types

Abstract the `Grid` class. The `getCell()` method's behavior will be at least partially defined in a subclass. Specifically, consider the following cases for a `getCell()` call:

- If `(i, j)` is within the grid, it returns 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 **the behavior will depend on the subclass**.
- If `(i, j)` is outside the bounding frame, throw an `OffTheGridException` to indicate that `(i, j)` is outside of any range that should ever be requested.

Create the following subclasses, in which the behavior of `getCell()` is fully defined:

1. `BoundedGrid`: Any access to the bounding frame returns a `ZombieCell`.
2. `WrapAroundGrid:` Access to a position on the bounding frame should *wrap around* to the opposite side of the grid. For example, in a grid of size $r \times c$, a request for a cell in row -1 should become a request to the cell in row $r - 1$. Likewise, a cell in column c should wrap around to be the cell in column 0.

1.3 Observe the user interface types

Although there is **no work for you to do here**, notice that `UserInterface` is **not** an abstract class, but rather an *interface*. The implementations of this interface, listed below, provide different ways to see the cells evolving. Check out this code to see what it's doing, and then try each one. Specifically, there are three implementations of `UserInterface` that you can choose when running `Life`:

1. `TextUserInterface`: Prints, as a log, the sequence of generations in rapid succession. You've seen this from the beginning of Project-1.

2. `SmartTextUserInterface`: Like the `TextUserInterface`, but it uses *Control Sequence Introducer (CSI) codes* to reprint the grid on top of itself in a *Terminal* window. 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.

2 Getting started

Gather the code: Start a *Terminal*. Then, grab some starting source code:

```
$ cd
$ curl -L https://bit.ly/cosc-112-24f-p2 -o project-2.zip
$ unzip project-2.zip
$ cd project-2
```

Next, make copies of some of your `Cell` classes from Lab-3, and then open it all:

```
$ cp ../lab-3/*Cell.java .
$ code .
```

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—that is, you are invited and encouraged to change existing classes. When done, a user should be able to run the program with their desired cell type, grid type, and user interface.

A special note about Morphing cells: Notice that when a Morphing cell changes into a Conway or Highlife cell, the grid at that position should **not contain a Morphing cell that merely acts like a Conway/Highlife cell**. Instead, an actual `ConwayCell` or `HighlifeCell` object should appear at that position in the grid once the morphing has occurred.

4 How to submit your work

Submit **all** of your `.java` source code files by uploading them into the `project-2` folder in your shared Google Drive folder for this course.

This assignment is due on Sunday, Oct-20, 11:59 pm.