1 Assignment

Summary: Write Java code to determine the state of a tic-tac-toe board and write tests that validate the correctness of your code. We will provide the function interface for your code, but the design of the implementation is up to you. Try to design and write your code to be as clean and readable as possible.

Background: Tic-tac-toe (`https://en.wikipedia.org/wiki/Tic-tac-toe`) is a paper-and-pencil game for two players, X and O, who take turns marking the spaces in a 3x3 grid with their mark. Player X is the first to make a mark. Once a space has been marked, it cannot be marked again. The player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal row wins the game, and the game is over (i.e., no further actions occur).

Specification: The provided function interface expects a description of the state of a tic-tac-toe board in the form of a Java `String`. This String should consist of 9 characters, one for each position of the board. The characters are ordered in groups of three characters (left-to-right) specifying top, middle, and then bottom horizontals, as shown in Figure 1. The String is case-insensitive. Squares marked by player X are specified with an 'x' or 'X' and squares marked by player O are specified with an 'o' or 'O'. Any other character is considered to be an empty square. Figure 1 shows what a board specified as "o-XxxoO.z" would look like.

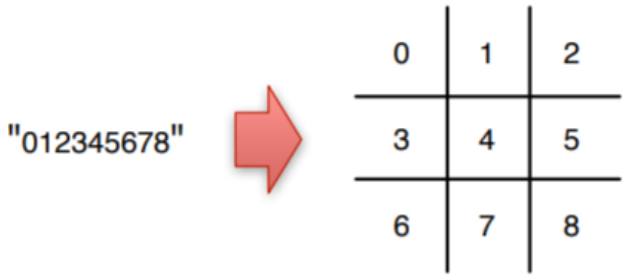

Figure 1. The mapping from Strings to positions in the Tic Tac Toe board.

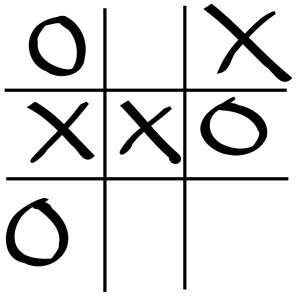

Figure 2. The above Tic Tac Toe board could be described as: o-XxxoO.z

Your implementation of `evaluateBoard` should analyze the state of the specified board and return one of five `Evaluation` values. The values are defined as a Java `enum` type in `Evaluation.java`. If you aren't familiar with `enum`s in Java, there are a lot of good references on the Internet; it isn't that difficult of a concept.

If the argument passed to the function doesn't correctly describe a board, your code should return

`InvalidInput`. If the argument describes a board state, but one that is not reachable by playing a game of tic-tac-toe, then your code should return `UnreachableState`. If the function's argument describes a valid, reachable board state, then it should return `Xwins`, `Owins`, or `NoWinner` if the board is in a state where X has won, where O has won, or where no one has won, respectively.

Your repository:

Use the following link to create your own copy of the TicTacToe repository on GitHub:

https://classroom.github.com/a/fEIephYg

In IntelliJ, create a new project using the following menu command:

```
File -> New -> Project From Version Control -> GitHub
```

A dialog should pop up. Select the appropriate repo from the list labeled "Git Repository URL". Click the "clone" button. Since the version of Java you are using might be different than the one I used to create the project, you might need change the **Project SDK** using the menu command below. Select a Project SDK that you have installed that is Java 1.7 or greater. For some of you a pop-up might appear asking you to choose a different Project SDK.

```
File -> Project Structure
```

To commit your work, use the following command. Write a useful commit message, and be sure to push your work to GitHub.

```
VCS -> Commit Changes
```

Be sure to frequently commit your changes. When writing code, you should break your work into lots of little pieces (15-60 minutes of effort). Your work cycle should be to: 1) pick the next piece that you can implement, 2) implement it, 3) debug it until you are confident that it works, and 4) commit and push it. That way, if anything happens, you never lose more than a small amount of work. **Part of your grade on code review assignments will be based on progressively committing your code.** Note: if you code is broken and not compiling, you probably want to get it working before committing.

Testing: The primary motivation for this assignment is to give you a moderately complicated function for which to write black box tests. Following a test-first philosophy, we're going to ask you to write your tests before writing your code. More precisely you should have tests checked in to the repository before you have code.

Your tests should be implemented in the provided `TicTacToeTest` class using Junit4. Each assert should be in a separate test. Name your tests with meaningful names. For most tests, no comments are necessary, because the name of the test is sufficiently explanatory. Write enough tests to give yourself confidence that your implementation will be correct if it passes all of the tests. Exhaustive

testing is untenable, as each of the nine positions of the board can be X, O, or blank, leading to 3^9 (over 19,000) board configurations.

To effectively test your code using a much smaller number of tests, use the "Bag of Tricks" we discussed in Lecture 2 (e.g., equivalence classes, boundary conditions, classes of bad data) to pick a useful set of inputs to test. Take an adversarial approach to writing your tests, trying to identify the problematic inputs that are likely to break your implementation and that of other students.

While testing is a key concept in this course and we expect tests where possible in all assignments in this particular assignment it is the focus. This means that in this assignment we consider your tests to be the same as your code. That is you should not share them with each other before code review.

Design and Style: We are using the Google Java Style Guide for the Java code that we write in CS 126. For this assignment, please read the portion of the style guide associated with naming (see the URL below) and follow the conventions described.

`https://google.github.io/styleguide/javaguide.html#s5-naming`