Introduction

In this assignment, we are going to implement the popular Tic-Tac-Toe game. The player will win when he/she gets three cells as a row, column, or diagonal. **In this version of the game, the board is by default 3x3 but can be of any number of rows and columns.** So your solution should be general for the size of the board (i.e., number of rows and columns), but for the number of cells to win, you can assume it is always 3.

So running the game as:

`java TicTacToe` ==> Run the game for 3x3 board (3 cells to win) java

`TicTacToe 4 5 3` ==> Run the game for 4x5 board (3 cells to win)

The players are indicating their next move from the command line as shown in the following example (where x always start to play):

```
$ java TicTacToe
   |   |
-----------
   |   |
-----------
   |   |

X to play:
```

Now X indicates which cell he/she wants to mark for his play.
Cells are numbered 1,2,3 .. 9 (if the board is 3x3) starting from the left side of the first row, moving right, then to the second row ... etc.
See the following example:

```
X to play:  5
   |   |
-----------
   | X |
-----------
   |   |

O to play:  2

   | O |
-----------
   | X |
-----------
   |   |

X to play: 1
 X | O |
-----------
   | X |
-----------
   |   |

O to play: 9
 X | O |
-----------
   | X |
-----------
   |   | O

X to play: 4
 X | O |
-----------
 X | X |
-----------
   |   | O

O to play: 6
 X | O |
-----------
 X | X | O
-----------
   |   | O

X to play: 7
 X | O |
-----------
 X | X | O
-----------
 X |   | O

Result: XWIN
 $
```

Implementation

We will only need five classes in this assignment. You cannot change any of the signatures of methods provided to you (**that is you cannot modify the methods at all**). You cannot add new publicmethods or variables. You can, however, add new private methods to improve the readability or the organization of your code.

Please implement the missing parts in each file based on the comments provided.

1) GameState.java

GameState is an enum type that is used to capture the current state of the game. Ithas four possible values:

- *PLAYING*: this game is ongoing,
- *DRAW*: this game is a draw,
- *XWIN*: this game as been won by the first player,
- *OWIN*: this game as been won by the second player.

This file is implemented and is ready.

Reference about enumeration:
https://docs.oracle.com/javase/tutorial/java/javaOO/enum.html

2) CellValue.java

CellValue is an enum type that is used to capture the state of a cell. It has three possible values:

- *EMPTY*: the cell is empty,
- *X*: there is a *X* in the cell,
- *O*: there is a *O* in the cell.

This file is partially implemented.

3) TicTacToeGame.java

Instances of the class *TicTacToeGame* represent a game being played. Each object stores the actual board, which is saved in a **single dimension array**. There is an instance method that can be used to play the next move. The object keeps track of the player's turn so that information is not specified: we simply specify the index to play and the object knows to play either an *X* or an *O*. The object also tracks the state of the game automatically.

The specification for our class *TicTacToeGame* is given in the provided zip file. You need to fill out all the missing parts, reading all the comments carefully before doing so. **You *cannot* modify the methods or the variables that are provided**. You can, however, add new *private*methods as required.

The template that you are working with contains the following:

- An instance variable is a reference to an array of CellValue to record thestate of the board.
- Some instance variables to record the game's number of columns and lines, the number of cells to align, the number of turns played ("level") and current state.
- Three constructors: the default one creates the usual game (the 3x3 grid, with a win being 3 similar cells aligned), a second one can be used to specify both lines and columns, and the last one is used to specify lines, columns, and a number of cells to align. As usual, all the instance variables must be initialized when the object is constructed.
- Getters for lines, columns, sizeWin, level and game's current state
- A method to query the object about the next player (that is, is it *X*'s or *O*'s turn to play?).
- A method play(int index) to play at a particular location in the game. This updates the game state and the board.
- We also have a helper method, private void setGameState(int index), which is used to compute and update the game state once a particular move is played in the method play.
- Finally, we have a method toString(), which returns a string representation of the current state of the board, as shown in the example before. An example of a String returned by toString would be, when printed out:

```
X to play: 4
 X | O |
-----------
 X | X |
-----------
   |   | O
```

There are a few situations that need our attention. For example, the index selected by the player may be invalid or illegal. In such cases, simply write an error message. The subsequent behavior of the method is unspecified, so simply implement something that seems to make sense.
One other situation would be that players continue the game after one of them wins. For testing purposes, we want that to be possible; however, the game state should reflect the first winner of the game. So if the players keep going after a win, a message is printed out, but the game continues as long as the moves are legal. The "first" winner remains.
Note that the method toString() returns a reference to a String; it is not actually printing anything. So that one String instance, when printed, should produce the expected output (in other words, that one string instance, when printed, will span several lines).

4) TicTacToe.java

This class implements playing the game. You are provided with the initial part, which creates the instance of the class TicTacToeGame based on the parameters submitted by the user. All you need to do here is implement the remainder of the main method, the part that plays the game. It basically loops through each step of the game until the gameis over. At each step, it displays the current game and prompts the next player, *X* or a *O*, for a cell to play. It then plays that cell and keeps going until the game is over, at which point it finishes (so although we said that TicTacToeGame is able to play past the winning point, that situation should never occur from the main of TicTacToe).
If the cell provided by the player is invalid or illegal, and a message is displayed to the user, who is asked to play again. Here are a couple of examples of this situation:

```
$ java TicTacToe
   |   |
-----------
   |   |
-----------
   |   |
___________

X to play: 2
   | X |
-----------
   |   |
-----------
   |   |

O to play: 10
The value should be between 1 and 9
   | X |
-----------
   |   |
-----------
   |   |

O to play: 2
This cell has already been played
   | X |
-----------
   |   |
-----------
   |   |
```