CSE 174: Fundamentals of Programming and Problem Solving
Program #11 – 26 Points – Due SUNDAY, 12/1/2013, by 11:59pm

Learning Outcomes:

- Be able to use 3rd party classes and objects, including those that represent strings, produce random numbers, perform math functions, format strings, and perform console input and output
- Understand and appropriately apply control structures, nested and un-nested
- Use console input/output in a program
- Determine necessary/desirable functions along with their need structure (parameters, return types, etc.)
- Read, analyze, and summarize computer programming resources including textbook, API documentation, and help forums

Grading Rubric:

	Full credit	No credit or Partial credit
Game correctness (12 pts)	The program plays a number of games of tic tac toe.	Program does not match specifications.
Method decomposition (8 pts)	Program logic was broken into multiple methods. No method has more than 30 lines of code.	Methods were too long, or did too much.
Use constants (2 pts)	Program makes appropriate use of the constants that are built in to the Cell and TicTacToe classes	There are places where constants should have been used, but weren't.
Format and Style (4 pts)	Code was written and commented to emphasize principles of readability, and to conform to our standard guidelines.	Errors/omissions in formatting and/or documentation.

Problem:

Write a class named `TicTacToeGame` that allows the user to play *any number* of tic tac toe games. The games may be *either*:

- human vs. human
- human vs. computer

(Or, as a bonus, you could offer both options, but do so in an elegant way...try to find a way to reuse methods as much as possible.)

Two classes are provided to support the implementation of the game, `TicTacToe` and `Cell`. You will need to download the source code for these two classes from the Documents section of the course website, and save them in the same folder where you save your own source code (no import statement is necessary to work with these classes because they will be located in the

same folder as your own file). You **MAY NOT** modify these files in any way. All the documentation you need for these files may be found at:

http://krumpe.com/javadoc/TicTacToe

Approach:

You are writing a third class, TicTacToeGame, that uses the two provided classes. Your code should create one TicTacToe object and use its methods. Whenever possible, you should use TicTacToe methods to assist in solving this problem. For example, you should not have nine variables that keep track of the tic-tac-toe grid, because the TicTacToe object keeps track of this for you. Similarly, TicTacToe objects have a method that will return a String representation of the tic-tac-toe grid. Use that method rather than writing one from scratch.

An empty game board appears as follows:

```
=============
|   |   |   |
=============
|   |   |   |
=============
|   |   |   |
=============
```

The first cell on the board (the upper left hand corner) is identified as (0,0) where the first number represents the row and the second number represents the column.

Whenever it is the user's turn:

- Show the user the board before asking the user to make a move.
- The user should enter a zero-based row and column for the move. For example, if the user wants to put a mark in the upper-left corner, she should enter **0 0**
- Your program should keep track of whose turn it is. Don't make the user enter the letter "X" or "O".
- Your program should use a loop to enforce rules about where a player moves. As long as the move is not valid, keep looping until it is valid. There are two ways that a move might not be valid:
    - The row and column are out of bounds (for example, **3 2** does not correspond to a square on the grid)
    - The row and column are already occupied (for example, **0 0** is not valid if there is already a move played in that location)

  The interaction for the user move could look something like this, for example:

```
Enter your move: 3 1
This is an illegal board position!
Enter your move: 0 0
This cell is occupied!
Enter your move: 2 1
etc...
```

- You ought to have a method for the human's turn. If two human's are playing, you ought to use the same method for both humans. That is, don't use one method for one human, but a different method for the other human.

Whenever it is the computer's turn (only applies if you are doing human vs. computer):

- Show the user the board before the computer makes a move.
- The computer moves should be random (you can use Math.random() for that purpose), but make sure that the computer doesn't make an invalid move.
- Print something to indicate where the computer plays...for example:

```
Computer plays 2 1
```

When the game ends, print something to indicate who wins. Show the board one last time.

After the game ends, ask if the user wants to play again.

Keep methods small...the smaller, and more "special purpose", the better.

No method should have more than 30 lines of code in its body. If it does, it's probably trying to do too much, and you should figure out a way to "off load" that logic to another method.

Take advantage of the constants that are built in to the Cell and TicTacToe classes.

There are constants in the Cell and TicTacToe classes. Here is a complete list of these constants:

```
Cell.EMPTY, Cell.X, Cell.O
TicTacToe.DRAW, TicTacToe.IN_PROGRESS,
TicTacToe.O_WINS, TicTacToe.X_WINS
```

You will quickly discover that these constants have pre-defined values. For example TicTacToe.XWINS has a value of 0. However, your code should NOT use those numeric values. In fact, I plan to change those numeric values in TicTacToe.java before run your code. So, if you use code like this, your code will still work:

```
if (myGame.gameStatus() == TicTacToe.X_WINS)
      System.out.println("X Wins!");
```

But if you code like this, your code will stop working correctly:

```
if (myGame.gameStatus() == 0)
      System.out.println("X Wins!");
```

Use constants whenever possible!

Turn-in:

Submit the source code for your `TicTacToeGame` class to our course website. Do not submit the other two classes.