Due Saturday by 11:59pm **Points** 2 **Available** Nov 13 at 10pm - Nov 19 at 11:59pm

Design of Model

Home (https://northeastern.instructure.com/courses/123246/pages/home)

Modules (https://northeastern.instructure.com/courses/123246/modules)

Lab 7: Tic-Tac-Toe Model

Starter Code: TicTacToe.java (https://northeastern.instructure.com/courses/123246/files/16688223?wrap=1) (https://northeastern.instructure.com/courses/123246/files/16688223/download?download_frd=1) TicTacToeModel.java (https://northeastern.instructure.com/courses/123246/files/16688222?wrap=1) (https://northeastern.instructure.com/courses/123246/files/16688222/download?download_frd=1) (https://northeastern.instructure.com/courses/123246/files/16688217?wrap=1) TicTacToeModelTest.java (https://northeastern.instructure.com/courses/123246/files/16688233?wrap=1) (https://northeastern.instructure.com/courses/123246/files/16688233/download?download_frd=1)

1.1 A Model for Tic Tac Toe

The purpose of this exercise is to give you practice with implementing the Model component of the Model-View-Controller design pattern.

In the starter code, you are given an interface representing a game of Tic Tac Toe; your task is to implement the TicTacToe interface.

You will need to define an enum Player, representing the players (`X` and `O`), with a toString() method that returns `"X"` and `"O"` accordingly. You will need to implement the public class named `TicTacToeModel`, with a single public constructor that takes no arguments. The class definition, with a `toString()` implementation to help with debugging, are provided to you in the starter code. You will fill in the fields and remaining method definitions as appropriate. You may also define other classes at your option as needed.

The game grid cells are numbered by row and column starting from 0. For example, the upper left position is row 0, column 0 (or `[0][0]` in the 2D array returned by `getBoard()`), the upper middle position is row 0, column 1 (`[0][1]`), the lower right is `[2][2]`.

1.2 Testing

We have supplied you with some basic JUnit tests as part of the starter code. Use these to verify that your implementation is correct, and write additional tests of your own.

1.3 Notes to Keep in Mind

- Avoid duplicating code as much as possible. Consider using non-public methods as means of creating reusable pieces of functionality.
- Be sure to use access modifiers, `private` and `public`, as well as `final`, appropriately.
- In your getters, be careful to return **a copy of**, and **not** a direct reference to, any mutable internal state in your model.
- Include JavaDoc for your classes and constructors as appropriate. You do not need to repeat JavaDoc already existing in a superclass or interface when you override a method.

1.4 To Turn In

Submit your zip containing only your src and test directories to the Lab assignment on the Gradescope server, or point your **private** GitHub repository to the submission.

NO UML is required. We've given you some basic unit tests - if you pass those, you'll receive full credit for this lab. Of course we encourage you to write more tests of your own.

Lab 7 - rubric

Full Credit - Pass	Partial Credit - Pass	No Credit
2 pts	**1 pt**	**0 pt**
Both solutions pass all automated tests.	Solutions pass 50% of the automated tests	Solutions pass < 50% of the automated tests. Or assignment not submitted