Homework 04 - Tic-Tac-Toe

Authors : Sumit Choudhury, Tejas Pradeep, Charlie King, Andrew Chafos, Vince Li, Julia Zhu, Shishir Bhat, Emma Barron

Purpose

This program is one of our first to use multiple classes and do more with the object-oriented part of "Object-Oriented Programming". We'll be breaking out of that static rut we've been in. This homework will give you experience coding some of the recently introduced concepts like arrays, static methods, instance data and methods, constructors, and toString.

For this homework, we'll be using two primary classes, `TicTacToe.java` and `Board.java`. You won't be writing all the code in these classes. (We were concerned that might be a bit much). Instead, we have provided some methods, and it's up to you to write the rest! As you code, be sure to think carefully about design. As programs grow in complexity, design becomes even more important, so be sure to follow good principles like encapsulation and modularity!

Background

Tic-Tac-Toe Rules

1. The game is played on a 3x3 grid.
2. Player 1 is the letter `X`, and Player 2 is the letter `O`. Players take turns putting their marks in empty squares.
3. The first player to get 3 marks in a row (vertically across any column, horizontally along any row, or diagonally across any diagonal) is the winner.
4. When all 9 squares are full, the game is over. If no player has won, the game ends in a tie.

Directions

Read the entire PDF and the provided code. Make sure you completely understand the assignment before beginning.

`GameState.java`

Provided Java file. It is an enum representing the game's state - a win for Player 1, a win for Player 2, a tie, or an ongoing game. **Do not modify this file.**

Location.java

This Java file represents a given location on the board. It has two instance variables: row and column. For example, a Location object where the row is 2 and the column is 2 would represent the square at the bottom right of a Tic-Tac-Toe board. Because they are private, we cannot access them from outside the Location class. Therefore, implement the following getter methods in `Location.java`. Make these methods public.

- a getter for row, properly named (ie, getRow)
- a getter for column, properly named (ie, getColumn)

`Board.java`

This Java file represents our Tic-Tac-Toe board. Create a 2D **character array** called `board` to hold the current state of the Tic-Tac-Toe board. This board should be 3x3 and be an **instance variable**. When the game begins, the `board` should only hold empty spots. An empty spot is represented with a space character (`' '`). As players place their tokens, it should begin to fill with `X`s and `O`s. Player 1 is `X` and Player 2 is `O`. You will implement the following public methods for `Board`.

- a method that takes in a location and returns a boolean.
  - True if the location is valid (ie, on the board) and the location is an empty spot.
  - False if the location is invalid or non-empty. An example of an invalid location would be the spot 40,2 because there is no row 40 in our 3x3 grid.
- a method that takes in a location and a letter, and returns a boolean
  - This method places the letter (which should be an `X` or `O`), in the specified location. This should only happen if the location is valid.
  - Return true if placing the letter is successful, false if not

We have provided the following public methods for `Board`.

- the `getGameState` method
  - Returns a GameState representing the current state of the game
  - Either there was a winner, a tie, or it's still ongoing
- the `toString` method
  - Returns a String representing the current game board
  - This might be very useful for printing out the current game board!

`TicTacToe.java`

We will run this Java file to play our game of Tic-Tac-Toe (i.e. this is our driver class). Implement the following methods. Make sure these methods are public!

- a `main` method.
  - Initialize a Scanner here
  - Call a method to prompt the user for the number of players
    - If there is one player, it will be a player vs computer game
    - If there are two players, it will be a player vs player game
    - Remember that Player 1 is always the letter `X`, and Player 2 is always the letter `O`.
  - Begin the game. Hint: Other methods in the TicTacToe class will be very useful here! Remember that we want to reuse code whenever possible.
  - Continue the game until someone has won or there is a tie. Once the game is over, the program will finish. It will not prompt the user if they would like to play again.
- a method which simulates a one-player game with the computer. Hint: Make sure you take advantage of methods in the Board and TicTacToe classes. They are handy and make this method much easier!
  - Create a Tic-Tac-Toe board.
  - While the game isn't over, the player and the computer will take turns placing their letter (`X` or `O`) on empty squares. Be sure to prompt the player for where they want to go.
  - The player will always be Player 1. The computer will always be Player 2.
  - On Player 1's turn, the program will print out the board. It will prompt the player for input and update the board with the player's move.
  - On the computer's turn, the computer will randomly pick an empty square on the board and place its letter there.
  - When the game is over, let the user know who won (or if it was a tie)
- a method which simulates a two-player game. Hint: Make sure you take advantage of methods in the Board and TicTacToe classes. They are handy and make this method much easier!
  - Create a Tic-Tac-Toe board.
  - While the game isn't over, prompt players for where they want to go. Players will take turns placing their letter (`X` or `O`) on empty squares
  - For each player's turn, the program will print out the board. Then it will prompt whoever's turn it is for input, and update the board with that player's move.
  - When the game is over, let the user know who won (or if it was a tie)

We have provided the following methods for `TicTacToe.java`.

- a `getNumberPlayers` method
  - Given a Scanner, this method prompts the user and returns the number of players.

- a `getInput` method
  - Given a Scanner, this method reads user input from the command line and returns a Location object.

Tips and Tricks

Reuse code whenever possible. Definitely check out the methods we provided - they are provided for a reason!

Create methods and variables as necessary. The ones listed above are just the ones we require.

Note: Make sure your classes are all in separate files and that all of your files are in the same directory.

Food for Thought

Why should the `board` variable of a `Board` object be an instance variable? Wouldn't this also work if it were static - ie, shared between all `Board` objects?

Well, imagine there were multiple games of Tic-Tac-Toe going on - perhaps your code was backing a website. If `board` were a static field, it would be shared among all instances of the class. All `Board` objects would have the same Tic-Tac-Toe board… despite users wanting to play separate games. That's not what we want! We want each `Board` to be separate.

Example Output:

```
How many players (1 or 2)? 1
  |   |
---------
  |   |
---------
  |   |

Enter desired square for 1: 1,1
  |   |
---------
  | X |
---------
  |   |

The Computer is making a move...
  |   |
---------
  | X |
---------
  | O |

Enter desired square for 1: 2,1
Enter desired square for 1: 1,1
Enter desired square for 1: 9,4
Enter desired square for 1: 2,2
  |   |
---------
  | X |
---------
```

```
  | O | X

The Computer is making a move...
  |   | O
---------
  | X |
---------
  | O | X

Enter desired square for 1: 0,0
X |   | O
---------
  | X |
---------
  | O | X

Player 1 wins!
```

Allowed Imports

To prevent trivialization of the assignment, you may only import `java.util.Scanner` and `java.util.Random`.

Checkstyle

You must run checkstyle on your submission. The checkstyle cap for this assignment is **15** points. If you don't have checkstyle yet, download it from Canvas -> Files/Resources. Place it in the same folder as the files you want checkstyled. Run checkstyle on your code like so:

```
$ java -jar checkstyle-8.28.jar yourFileName.java
Starting audit...
Audit done.
```

The message above means there were no Checkstyle errors. If you had any errors, they would show up above this message, and the number at the end would be the points we would take off (limited by the checkstyle cap mentioned above). The Java source files we provide contain no Checkstyle errors. In future homeworks we will be increasing this cap, so get into the habit of fixing these style errors early!

Feature Restrictions

There are a few features and methods in Java that overly simplify the concepts we are trying to teach or breaks our auto grader. For that reason, do not use any of the following in your final submission: var (the reserved keyword) System.exit

Collaboration

Collaboration Statement

To ensure that you acknowledge collaboration and give credit where credit is due, **we require that you place a collaboration statement as a comment at the top of at least one java file that you submit**. That collaboration statement should say either:

I worked on the homework assignment alone, using only course materials.

or

In order to help learn course concepts, I worked on the homework with [give the names of the people you worked with], discussed homework topics and issues with [provide names of people], and/or consulted related material that can be found at [cite any other materials not provided as course materials for CS 1331 that assisted your learning].

Recall that comments are special lines in Java that begin with `//`.

Turn-In Procedure

Submission

To submit, upload the files listed below to the corresponding assignment on Gradescope:

- `TicTacToe.java`
- `Board.java`
- `Location.java`
- `GameState.java`

Make sure you see the message stating "HW04 submitted successfully". From this point, Gradescope will run a basic autograder on your submission as discussed in the next section.

You can submit as many times as you want before the deadline, so feel free to resubmit as you make substantial progress on the homework. We will only grade your last submission: be sure to **submit every file each time you resubmit**.

Gradescope Autograder

For each submission, you will be able to see the results of a few basic test cases on your code. Each test typically corresponds to a rubric item, and the score returned represents the performance of your code on those rubric items only. If you fail a test, you can look at the output to determine what went wrong and resubmit once you have fixed the issue.

The Gradescope tests serve two main purposes:

1. Prevent upload mistakes (e.g. forgetting checkstyle, non-compiling code)
2. Provide basic formatting and usage validation

In other words, the test cases on Gradescope are by no means comprehensive. Be sure to thoroughly test your code by considering edge cases and writing your own test files. You also should avoid using Gradescope to compile, run, or checkstyle your code; you can do that locally on your machine.

Other portions of your assignment can also be graded by a TA once the submission deadline has passed, so the output on Gradescope may not necessarily reflect your grade for the assignment.

Important Notes (Don't Skip)

- Non-compiling files will receive a 0 for all associated rubric items
- Do not submit `.class` files.
- Test your code in addition to the basic checks on Gradescope
- Submit every file each time you resubmit
- Read the "Allowed Imports" and "Restricted Features" to avoid losing points
- Check on Piazza for all official clarifications
