Homework 3

Julia Ting

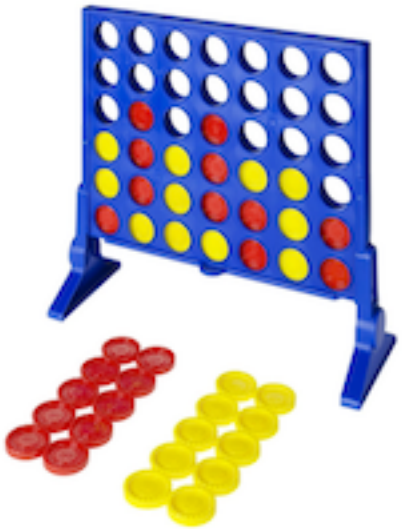
1 Introduction

Hello, hello, hello!

Since we're all masters of command line input and output, it's now time to move on to mastering arrays and conditional flow of logic. What you will be doing here is creating your own Connect Four game that can be played through the command line. Wooohoo! As always, please read *the entire document* as there are things even at the very end that you will not want to miss for this and future assignments!

2 Connect Four

Connect Four uses a 7 column, 6 row vertically suspended grid. Players take turns dropping colored discs into these slots. The goal of the game is to connect four of your own color discs either horizontally, vertically, or diagonally. If you haven't played it before, you can familiarize yourself with it here. A board in real life looks something like the following.



3 Problem Description

Your job will to be implement a version of Connect Four that you can play via the command line. In this version, the user can choose to play against a computer (single player mode) or against another player (two player mode). The mode of the game is specified via a command line argument. To give you a better idea of how this will work, the following examples have been given.

- Running the program with command line arguments. A 2 here specifies 2 player mode. A 1 here specifies 1 player mode.

```
java ConnectFour 2

Welcome to 2-player mode of Connect Four!
Choose where to go by entering the slot number.
Type 'q' if you would like to quit.

|    |    |    |    |    |    |    |
|    |    |    |    |    |    |    |
|    |    |    |    |    |    |    |
|    |    |    |    |    |    |    |
|    |    |    |    |    |    |    |
|    |    |    |    |    |    |    |
---------------------------------
  1    2    3    4    5    6    7

Player 1, where would you like to go?
4
Choose where to go by entering the slot number.
Type 'q' if you would like to quit.

|    |    |    |    |    |    |    |
|    |    |    |    |    |    |    |
|    |    |    |    |    |    |    |
|    |    |    |    |    |    |    |
|    |    |    |    |    |    |    |
|    |    |    | X  |    |    |    |
---------------------------------
  1    2    3    4    5    6    7

Player 2, where would you like to go?
4
Choose where to go by entering the slot number.
Type 'q' if you would like to quit.

|    |    |    |    |    |    |    |
|    |    |    |    |    |    |    |
|    |    |    |    |    |    |    |
|    |    |    |    |    |    |    |
|    |    |    | O  |    |    |    |
|    |    |    | X  |    |    |    |
---------------------------------
  1    2    3    4    5    6    7
Player 1, where would you like to go?
3
Choose where to go by entering the slot number.
Type 'q' if you would like to quit.

|    |    |    |    |    |    |    |
|    |    |    |    |    |    |    |
|    |    |    |    |    |    |    |
|    |    |    |    |    |    |    |
|    |    |    | O  |    |    |    |
|    |    | X  | X  |    |    |    |
---------------------------------
  1    2    3    4    5    6    7
```

- Running the program without command line arguments

```
java ConnectFour

Would you like to play single player or two player mode? (Enter 1 or 2)
2
Welcome to 2-player mode of Connect Four!
Choose where to go by entering the slot number.
Type 'q' if you would like to quit.

|    |    |    |    |    |    |    |
|    |    |    |    |    |    |    |
|    |    |    |    |    |    |    |
|    |    |    |    |    |    |    |
|    |    |    |    |    |    |    |
|    |    |    |    |    |    |    |
---------------------------------
  1    2    3    4    5    6    7
```

- Output when a user quits

```
Choose where to go by entering the slot number.
Type 'q' if you would like to quit.

|    |    |    |    |    |    |    |
|    |    |    |    |    |    |    |
|    |    |    |    |    |    |    |
|    |    |    |    |    |    |    |
|    |    |    | O  |    |    |    |
|    |    |    | X  |    |    |    |
---------------------------------
  1    2    3    4    5    6    7
Player 1, where would you like to go?
q
|    |    |    |    |    |    |    |
|    |    |    |    |    |    |    |
|    |    |    |    |    |    |    |
|    |    |    |    |    |    |    |
|    |    |    | O  |    |    |    |
|    |    |    | X  |    |    |    |
---------------------------------
  1    2    3    4    5    6    7
Goodbye!
```

- Output when a winner is found

```
Choose where to go by entering the slot number.
Type 'q' if you would like to quit.

|    |    |    |    |    |    |    |
|    |    |    |    |    |    |    |
|    |    |    | X  |    |    |    |
|    |    |    | O  |    |    |    |
|    | X  | O  | O  | O  |    |    |
| X  | O  | X  | X  | O  |    |    |
---------------------------------
  1    2    3    4    5    6    7
Player 1, where would you like to go?
3
|    |    |    |    |    |    |    |
|    |    |    |    |    |    |    |
|    |    |    | X  |    |    |    |
|    |    | X  | O  |    |    |    |
|    | X  | O  | O  | O  |    |    |
```

```
| X | O | X | X | O |   |   |
---------------------------------
  1   2   3   4   5   6   7
Woohoo! Player 1 won!
```

- Output when no winner is found

```
Choose where to go by entering the slot number.
Type 'q' if you would like to quit.

| O | O | X | X | O |   | O |
| X | X | O | X | O | X | X |
| O | X | O | X | O | O | O |
| X | O | O | O | X | O | O |
| O | X | X | O | O | X | X |
| X | O | X | X | O | X | O |
---------------------------------
  1   2   3   4   5   6   7
Player 1, where would you like to go?
6
| O | O | X | X | O | X | O |
| X | X | O | X | O | X | X |
| O | X | O | X | O | O | O |
| X | O | O | O | X | O | O |
| O | X | X | O | O | X | X |
| X | O | X | X | O | X | O |
---------------------------------
  1   2   3   4   5   6   7
There was a tie!
```

- Output when a user chooses a slot that is full (works repeatedly). The user should still be able to type "q" to quit!

```
Choose where to go by entering the slot number.
Type 'q' if you would like to quit.
|   |   |   | O |   |   |   |
|   |   |   | X |   |   |   |
|   |   |   | O |   |   |   |
|   |   |   | X |   |   |   |
|   |   |   | O |   |   |   |
|   |   |   | X |   |   |   |
---------------------------------
  1   2   3   4   5   6   7
Player 1, where would you like to go?
4
That column is full. Please pick another!
Player 1, where would you like to go?
4
That column is full. Please pick another!
Player 1, where would you like to go?
3
Choose where to go by entering the slot number.
Type 'q' if you would like to quit.
|   |   |   | O |   |   |   |
|   |   |   | X |   |   |   |
|   |   |   | O |   |   |   |
|   |   |   | X |   |   |   |
|   |   |   | O |   |   |   |
|   |   | X | X |   |   |   |
---------------------------------
  1   2   3   4   5   6   7
Player 2, where would you like to go?
4
That column is full. Please pick another!
```

```
Player 2, where would you like to go?
q
|    |    |    | O |    |    |    |
|    |    |    | X |    |    |    |
|    |    |    | O |    |    |    |
|    |    |    | X |    |    |    |
|    |    |    | O |    |    |    |
|    |    | X  | X |    |    |    |
---------------------------------
  1    2    3    4    5    6    7
Goodbye!
```

4 Solution Description

4.1 Basic Game Play

The actual Connect Four board is represented by a two dimensional String array. Whenever a player goes, you should insert the appropriate token into the appropriate spot in the 2D array.

Your game play must have the following functionality. After reading the below requirements, be SURE to visit the Important Notes, Tips, and Tricks section as there is indeed additional important information.

1. **Your game should have two modes: 1 player and 2 player.** The mode should be determined by a command line argument, either a 1 or a 2. If the user does not specify a command line argument, you should prompt them through the command line.

   (a) For 1 player mode, you will be creating your very own connect four AI! This AI will be very dumb because it will simply be choosing moves via a random number generator.

   - Your AI should generate moves until a valid one is found. This means if the first number it generates is a column that is already full, you must continue to randomly generate until you find a column that is playable.
   - Draw upon our discussion of Random in class, and of course, you can always look it up in the API. This is different from Math.random(). If you use Math.random() in this assignment, you will not receive points for your implementation of the AI. See the Tips and Tricks section if you are still unsure how to go about this.

2. **The player(s) should specify which slot they want to "drop" their disc in by entering the corresponding slot number.** Slot numbers start at 1 and go from left to right. *NOTE: This is different from array indices, which start at 0!*

3. **After a move has occurred, you must print out the Connect Four board to reflect the appropriate move.** For this game Player 1 will always be X and Player 2 (or the computer) will always be O. We have provided String variables called `TOKEN1` and `TOKEN2` for you to use.

   (a) Your board must print the associated slot numbers in a legible manner.

   (b) Your board must also be aligned properly, so the user can easily tell which column is which.

(c) *NOTE: In following these two requirements, your board format should essentially match the example output above, give or take a few spaces.*

4. **Once the game ends, you must specify the result.** We have provided for you an enum `GameResult` with all the potential outcomes. We have also provided for you a method `findWinner()` that will return whether or not a winner exists with the current state of the board. A more detailed description of what findWinner() does can be found in section 4.3.

   (a) After the game ends, make sure you print the board out a final time so everyone can see what the final board looks like!

   (b) **If a user chooses to quit**, you must print out some message acknowledging that the user has quit successfully.

4.2 Handling Full Columns

As you may have noticed in the example output, you will also need to be able to handle when a user selects an unplayable column. This means that if a slot is already filled to the top, a user shouldn't be able to put another token in that slot. You should ensure any attempt to do this doesn't crash your program! Since we are very forgiving programmers and gamers, you should allow the user to keep picking until they select a column that is playable. Additionally, **the player should still be able to type "q" to quit the program** during this reprompting.

NOTE: You should never encounter a situation where all slots are unplayable and the game hasn't ended. if you do, make sure you are calling findWinner() correctly as that method handles the situation in which all possible moves are exhausted.

4.3 findWinner()

`findWinner()` is a method that returns a GameResult enumeration, as mentioned above. Here are the different possible return values and the corresponding situations.

1. **`findWinner()` returns `GameStatus.ONE`**. This means Player 1 has won.
2. **`findWinner()` returns `GameStatus.TWO`**. This means Player 2 has won.
3. **`findWinner()` returns `GameStatus.TIE`**. This means all possible moves have been exhausted and neither Player 1 nor Player 2 have won.
4. **`findWinner()` returns `GameStatus.ONGOING`**. This means currently there is no winner, nor is there a tie. Game play should continue if this is returned.

There is another value for `GameStatus` not returned by `findWinner()`, `GameStatus.QUIT`. You will still want to use this value!

4.4 Implementation Details

There are two locations in which you will need to write your code.

1. **The main method**. This is where you will implement the majority of your functionality. This includes likely some looping mechanism that will continue until an outcome has been achieved. You should also be calling `findWinner()` here!

   (a) You must use the provided variable `String[][] board` for your Connect Four board! Note you will also be using this variable in your printBoard() method.

2. **printBoard()**. This is where you will write the print functionality of your Connect Four board. It is your job to call this method within the main method each time you want to print the board!

4.5 Important Notes, Tips, and Tricks

- Checkstyle will count for this homework! **This means you will be docked points for checkstyle errors found in your homework submission.**
- You are NOT permitted to used any other data structure in the place of arrays. Additionally, you are NOT permitted to use any class that trivializes this assignment. This includes but is not limited to `Arrays.java`, `ArrayList.java`, etc.
- For the purpose of this assignment, you can assume the user will only ever input "q" or valid column numbers (1-7). This doesn't mean the slot will always be valid, as we know in the case of unplayable columns.
- If your program crashes with correct input, it will be **-25 points**.
- Arrays are indexed starting left to right and top to bottom. Think about how this might affect how you fill in your array when a user picks a given slot.
- You will need to keep track the row that the next token will be placed in per column. A good way to do this is by using another array!
- There will be lots of looping during this homework. A good way to start is to just write code for one iteration and THEN add in the looping aspect. This will simplify the debugging process on your loop conditions.
- We have given you several helpful variables. Make sure you think about how using these variables can be utilized in your program! They may help simplify your logic.
- You will be using `java.util.Random` and some of its various methods. Whenever you want to use the `nextInt()` method, use the method by the same name which takes in a bound: `nextInt(int bound)`. Read the API as to what numbers are generated and think about what number you might want to use as your bound.
- Feel free to consult outside sources for help! The TAs are always here to assist, and a quick Google search for some topic or concept can be beneficial as well.
- For those of you that like to procrastinate, don't. This homework might be a little tricky even though it doesn't sound like it. I'd hate for you to realize that on the day it is due.

5 Checkstyle

Checkstyle counts for this homework. You may be deducted up to 10 points for having Checkstyle errors in your assignment. Each error found by Checkstyle is worth one point. This cap will be raised next assignment. Again, the full style guide for this course that you must adhere to can be found by clicking **here**.

Come to us in office hours or post on Piazza if you have specific questions about what Checkstyle is looking for and how to fix Checkstyle errors.

First, make sure you download the `checkstyle-6.0-all.jar` and `cs1331-checkstyle.xml` from the T-Square assignment page. Then, make sure you put *both* of those files in the same directory (folder) as the `.java` files you want to run Checkstyle on. Finally, to run Checkstyle, type the first line into your terminal while in the directory of your Java files and press enter.

```
$ java -jar checkstyle-6.2.2.jar -c cs1331-checkstyle.xml *.java
Starting audit...
Audit done.
```

The message above means there were no Checkstyle errors. You can easily count Checkstyle errors by piping the output of Checkstyle through `wc -l` and subtracting 2 for the two non-error lines printed above (which is how we will deduct points). For example:

```
$ java -jar checkstyle-6.2.2.jar -c cs1331-checkstyle.xml *.java | wc -l
      2
```

Alternatively, if you are on Windows, you can use the following instead:

```
C:\> java -jar checkstyle-6.2.2.jar -c cs1331-checkstyle.xml *.java | findstr /v "Starting
    audit..." | findstr /v "Audit done" | find /c /v "hashcode()"
0
```

6 Turn-in Procedure

Submit your `ConnectFour.java` file on T-Square as an attachment. Do not submit any compiled bytecode (`.class` files), the Checkstyle jar file, or the `cs1331-checkstyle.xml` file. When you're ready, double- check that you have submitted and not just saved a draft.

Keep in mind, unless you are told otherwise, non-compiling submissions on ALL homeworks will be an automatic 0. You may not be warned of this in the future, and so you should consider this as your one and only formal warning. This policy applies to (but is not limited to) the following:

1. Forgetting to submit a file
2. One file out of many not compiling, thus causing the entire project not to compile
3. One single missing semi-colon you accidentally removed when fixing Checkstyle stuff
4. Files that compile in an IDE but not in the command line
5. Typos

6. etc...

We know it's a little heavy-handed, but we do this to keep everyone on an even playing field and keep our grading process going smoothly. So please please make sure your code compiles before submitting. We'd hate to see all your hard work go to waste!

7 Verify the Success of Your Submission to T-Square

Practice safe submission! Verify that your HW files were truly submitted correctly, the upload was successful, and that the files compile and run. It is solely your responsibility to turn in your homework and practice this safe submission safeguard.

1. After uploading the files to T-Square you should receive an email from T-Square listing the names of the files that were uploaded and received. If you do not get the confirmation email almost immediately, something is wrong with your HW submission and/or your email. Even receiving the email does not guarantee that you turned in exactly what you intended.

2. After submitting the files to T-Square, return to the Assignment menu option and this homework. It should show the submitted files.

3. Download copies of your submitted files from the T-Square Assignment page placing them in a new folder.

4. Recompile and test those exact files.

5. This helps guard against a few things.

   (a) It helps insure that you turn in the correct files.

   (b) It helps you realize if you omit a file or files.[1] (If you do discover that you omitted a file, submit all of your files again, not just the missing one.)

   (c) Helps find last minute causes of files not compiling and/or running.

[1] Missing files will not be given any credit, and non-compiling homework solutions will receive zero points. Also recall that late homework will not be accepted regardless of excuse. Treat the due date with respect. The real due date is 8PM Thursday. Do not wait until the last minute!