

WES-CS GROUP MEETING #12

Exercise 1: Board Games (Two-D Arrays)

For many board games (e.g., tic-tac-toe, chess, checkers, go) the board can be represented using a 2-dimensional array of characters, where the value in position `[j][k]` tells what piece (if any) is currently at that position.

Part (a)

An important method that a board game should supply is one that draws the current board with horizontal and vertical lines separating the squares. For example, a game's `draw` method might produce output like this:

```
+-----+
| x |   | o |
+-----+
|   | x |   |
+-----+
| o | x |   |
+-----+
```

Write a `BoardGame` class with a constructor and a `draw` method. The constructor should have two parameters: the number of rows and the number of columns. It should create a board then fill it randomly with X's, O's, and empty spaces. Write a `PlayGame` class that creates and prints a board game. Test your code then swap with another group and test their code.

Part (b)

For some games (like tic-tac-toe), deciding whether a player has won the game involves looking for certain patterns on the board. For example, for tic-tac-toe, to see if the X player has won we'd want to look for three X's in a row across, down, or on either diagonal.

To do that, you could add the following (incomplete) methods to your BoardGame class:

```
/** public methods */
public boolean hasWon(char playerChar) {
    return (winAcross(playerChar) ||
            winDown(playerChar) ||
            winDiag(playerChar));
}

/** private methods */
private boolean winAcross(char playerChar) { ... }

private boolean winDown(char playerChar) { ... }

private boolean winDiag(char playerChar) { ... }
}
```

Write the `hasWon`, `winAcross`, `winDown` and `winDiag` methods, assuming that a winning configuration is three in a row. Be sure to test your code. If you finish early, change your methods to take a second parameter: the number of characters in a row that constitute a win.

Exercise 2: Connect-4

One board game that fits the pattern we've been talking about is Connect-4 (it uses a board that can be represented using a 2D array, and a win for one player is four pieces in a row). Take some time now to play the game a few times with a partner. Think about how to write a program that allows two people to play Connect-4 against each other. What methods besides `draw` and `hasWon` might you want?

Connect-4 is also good for exercising your logical thinking muscles! Think about strategies that prevent your opponent from getting four in a row, and also some strategies for getting four in a row yourself.

Exercise 3: Programming Connect-4

First try out the Connect-4 program that's on the laptops. Are there any aspects that you think could be improved?

Then work together in groups to design your own version of the `Connect4` class (based on the `BoardGame` class you wrote previously) as well as a `PlayGame` class.

What fields and public methods should the `Connect4` class provide? What code belongs in the `PlayGame` class?

When you've finished your design, discuss it with the other groups. Then go ahead and write the code.