Method Exercise - Noughts & Crosses/Tic-Tac-Toe

This exercise sheet only has one ‘big’ problem and it involves writing more than one method.

The point of this exercise is to write a version of the game *Noughts & Crosses*, commonly also called *Tic-Tac-Toe*. If you don’t know the game:

https://en.wikipedia.org/wiki/Tic-tac-toe

This looks intimidating and to be fair it’s quite complex - but it’s important to start to build more complex programs with more than one method.

Super Challenge

Just write an implementation of the game without reading the next bit. Then go on, if you’re feeling confident, and do the Mega Challenge at the end.

The point of this game, like LastBiscuit, is for two human players to play each other. The problem divides into a number of parts - there are some hints here about what the parts are, and what they should do, but writing them (and putting them together) is up to you.

Representing the Game

You need some way of representing the state of play - it needs a 3x3 board and you have to be able to represent squares that are empty, contain a cross or a circle. Maybe something like this:

```
char[][] ticTacToeBoard = {
    {' ', ' ', ' '},
    {' ', ' ', ' '},
    {' ', ' ', ' '}
};
```

Which is a 2D array (3x3) of blank spaces. The game will replace the space characters with ‘x’ and ‘o’ as it proceeds. Or you could use integers (maybe -1 for O and 1 for X, with 0 for blank - or similar).

Challenge

Define a three-valued *enumerated type* for the possible states of a square on the board instead of an int or char.

Printing the Board

You need a method to print out the board. For example:

```
x|  |x
——–
o|  |
——–
o|  |
```

is the game after two rounds (and hopefully it’s ‘o’s go - or they’ve just lost).

Telling if There's a Winner

You need a method to tell if the game has been won by someone - maybe:

```
static boolean hasWon(char[][] board, char player)
```

This method checks to see if the player represented by a particular character (X or O, or whatever you have chosen) has won, and if so returns true - you could call it twice per round to see a particular player has won (or more efficiently you could call it once after each player has had their turn).

This might seem a tricky one to do, and to be fair it can be a bit tedious, but there are only 8 possible winning patterns to consider. For example:

```
boolean topRowWin = board[0][0] == player &&
                    board[0][1] == player &&
                    board[0][2] == player;
```

is a boolean which is true if the top row is all set to the same character, represented by player. You can define these for all 8 possibilities and return true if one of them is true.

Getting Player Input

You need a method to get the row/column number of the square the player wants to choose. This should just ask for a number between 1 and 3 twice, once for row and once for column.

Slight Challenge Version

Extend your input method so it checks to see the square has not been played already.

Update the Board

Once you have the position a player has chosen, update the board - set the relevant square to 'x' or 'o' (or whatever value you've chosen), depending on which player they are.

Put them Together

Write a main method including a loop, that calls your methods and continues until one of the two players wins - print out the winner.

Mega Challenge

Extend your program so that the computer plays instead of two humans. The 'easy' version of this is to choose a random position (making sure it's not taken). The 'hard' (potentially 'very hard') way is to implement a strategy - there's one on the Wikipedia page above if you scroll down. You can simplify it if you want - the Wikipedia strategy guarantees you at least draw but you could go for something less good. (And of course there's also:

http://xkcd.com/832/

Which could be handy as a human player but probably not a lot of use to you programming it.)