ASSIGNMENT 2
A Game of Connect Three

COMP-202, Summer 2010

Due: Thursday, May 20th, 2010

In this assignment we will program our first little game. Some of the more simple games out there are implementations of board games, without computer opponents. We will implement connect three, which is just a smaller version of connect four, on a 4x3 grid. It is a two-player game in which the players take alternating turns dropping pieces down this grid, vertically down, which fall on the next available space in the same column. The player who first connects three of her pieces vertically, horizontally or diagonally, wins. If after 12 turns no player wins, there is a draw.

You should follow the below specification very closely. If you read the instructions carefully, it will make the assignment easier to complete. A specification like this is basically a checklist, even if it is written as a block of text.

Do the assignment in the below order, and test every method you write. We will not only test the whole program, but also the correctness of every single method.

1 Connect Three

1.1 `b11..b34`

Create a class named `ConnectThree` (in a file called ConnectThree.java), In this class define twelve static char's `b11-b14, b21-b24, b31-b34` and initialize them with `' '`. These variables represent the 12 pieces of the board. An empty position will be represented by `' '`, a piece of player 1 by `'X'`, and of player 2 by `'O'`. We will also represent the players themselves by `'X'` and `'O'`, using characters. This will make it easier because the board logic and the printing of the board will use the same characters.

1.2 `printBoard`

Within the `ConnectThree` class, create a parameterless method called `printBoard`. This method should print out the current board (using the b_{ij} variables) in a readable way, with the numbers representing the columns below. Refer to the example output for the desired formatting. Test the correctness of your code with a main method which sets a couple of pieces.

1.3 `placePiece`

Create a method `placePiece`, taking two parameters. The first is an integer denoting the location where a piece is getting dropped. The second is a character denoting the player which dropped a piece. The location should be between 1 and 4, and the player should either be `'X'` or `'O'`. The method should figure out where the piece has to go, by looking at the column of the b_{ij} variables denoted by the location parameter and finding the 'lowest' position which is still empty (i.e. there is a `' '` there), and set the corresponding b_{ij} variable to the value of the player parameter. If the location variable is not between 1 and 4, or there is no free piece in that column, then the method should do nothing (i.e. the player forfeits the turn). Hint: you will need a nested if for the location (the column), and then for checking the position (the row) where to drop the piece. Change the testing code in your main method to call placePiece a couple of times, verifying correctness with your printBoard method.

1.4 `checkWinAndExit`

Create a method called `checkWinAndExit`, taking a character as a parameter denoting a player (`'X'` or `'O'`). The method should check whether the given player has won the game, i.e. whether the player has connected three of her pieces. If the player won, the method should output that the player (including the player character) won and exit the program using `System.exit(0)`. Hint:

- There are 6 horizontal winning possibilities (2 per row), 4 vertical winning possibilities, and 4 diagonal winning possibilities.
- You can check one winning possibility with a single expression checking whether all three pieces equal the player parameter. You can also group these expressions together, for example by noting that all solutions in one row need the two middle pieces.
- It is probably easiest to create a temporary boolean variable (i.e. called `won`). You set it to true if you find a winning combination. And at the end of the method you print and exit if the `won` variable is true.

Test your method again by modifying the main method

1.5 `makeTurn`

Now create a method called `makeTurn`, which takes one parameter. This parameter is a character denoting a player to make a turn (either `'X'` or `'O'`). The method should first prompt for, then let the user input a location where to drop a piece. It should then place the piece with the `placePiece` method (note how a player forfeits the turn for invalid inputs because they get ignored), and print the board. The method should then check if the current player won (which will exit the program if she did).

1.6 `main`

Now we can put it all together using the main method. The main method will first print out the board and then make the 12 alternating turns using the `makeTurn` method (starting with the `'X'` player). If after the 12 turns no player has won, The program should output that there was a draw.

2 Optional bonus worth 15% extra

As an optional bonus, for a total of 115%, you may implement connect four on a 6x4 board. You have to implement the connect three program, and you can change it for a larger board and the necessity to connect 4 pieces. Call the new class `ConnectFour`. If you would like to attempt the bonus question, you have to submit both `ConnectThree.java` and `ConnectFour.java`.

Do's and Don'ts:

- Use file names, class name, method and global variable names exactly as specified.
- Put your name and student ID at the top of every file as a comment
- Give all variables useful/descriptive names according to the conventions, and follow the conventions on indentation and commenting.
- Submit your java file (just the source file) via WebCT.
- Use only java features that have been covered in class.
- You may discuss the assignment with other students, put do not share any code. All submitted work has to be your own.

Example runs

```
>> java ConnectThree

    |       |
    |       |
    |       |
 -----------------
     1 2 3 4
X's turn, where to place: 1

    |       |
    |       |
    |X      |
 -----------------
     1 2 3 4
O's turn, where to place: 2

    |       |
    |       |
    |X O    |
 -----------------
     1 2 3 4
X's turn, where to place: 2

    |       |
    |  X    |
    |X O    |
 -----------------
     1 2 3 4
O's turn, where to place: 3

    |       |
    |  X    |
    |X O O  |
 -----------------
     1 2 3 4
X's turn, where to place: 4

    |       |
    |  X    |
    |X O O X|
 -----------------
     1 2 3 4
O's turn, where to place: 3

    |       |
    |  X O  |
    |X O O X|
 -----------------
     1 2 3 4
X's turn, where to place: 3

    |    X  |
    |  X O  |
    |X O O X|
 -----------------
     1 2 3 4
Player X won!
```

```
>> java ConnectThree

    |       |
    |       |
    |       |
 -----------------
     1 2 3 4
X's turn, where to place: 1

    |       |
    |       |
    |X      |
 -----------------
     1 2 3 4
O's turn, where to place: 1

    |       |
    |O      |
    |X      |
 -----------------
     1 2 3 4
X's turn, where to place: 2

    |       |
    |O      |
    |X X    |
 -----------------
     1 2 3 4
O's turn, where to place: 2

    |       |
    |O O    |
    |X X    |
 -----------------
     1 2 3 4
X's turn, where to place: 1

    |X      |
    |O O    |
    |X X    |
 -----------------
     1 2 3 4
O's turn, where to place: 3

    |X      |
    |O O    |
    |X X O  |
 -----------------
     1 2 3 4
X's turn, where to place: 2

    |X X    |
    |O O    |
    |X X O  |
 -----------------
     1 2 3 4
O's turn, where to place: 4
```

```
    |X X    |
    |O O    |
    |X X O O|
 -----------------
     1 2 3 4
X's turn, where to place: 3

    |X X    |
    |O O X  |
    |X X O O|
 -----------------
     1 2 3 4
O's turn, where to place: 3

    |X X O  |
    |O O X  |
    |X X O O|
 -----------------
     1 2 3 4
X's turn, where to place: 4

    |X X O  |
    |O O X X|
    |X X O O|
 -----------------
     1 2 3 4
O's turn, where to place: 4

    |X X O O|
    |O O X X|
    |X X O O|
 -----------------
     1 2 3 4
There was a draw!
```