# Programming IT 2019-2020

Assessed Exercise 2

Dr. Simon Rogers & Prof. Alice Miller

## Connect 4

Connect 4 is a popular game played by two **players**. It consists of a **board** which is made up of X **columns,** into which **counters** can be dropped. When dropped, they come to rest at the bottom of the column (if it is empty), or on top of the last counter to be dropped into that column if it is not empty. Each column can contain a maximum of Y counters.
Each player uses counters of a different colour.
The objective of the game is to place your counters such that they make a line of four. A line can be horizontal, vertical, or diagonal. For example, in the board depicted below (with X = 7 columns numbered 0 to 6, and Y = 6 rows), if it were the red player's turn, dropping a counter into column 4 would win them the game (creating a diagonal line of 4). If it were the yellow player's turn, dropping a counter into column 1 would win them the game (creating a vertical line of 4).

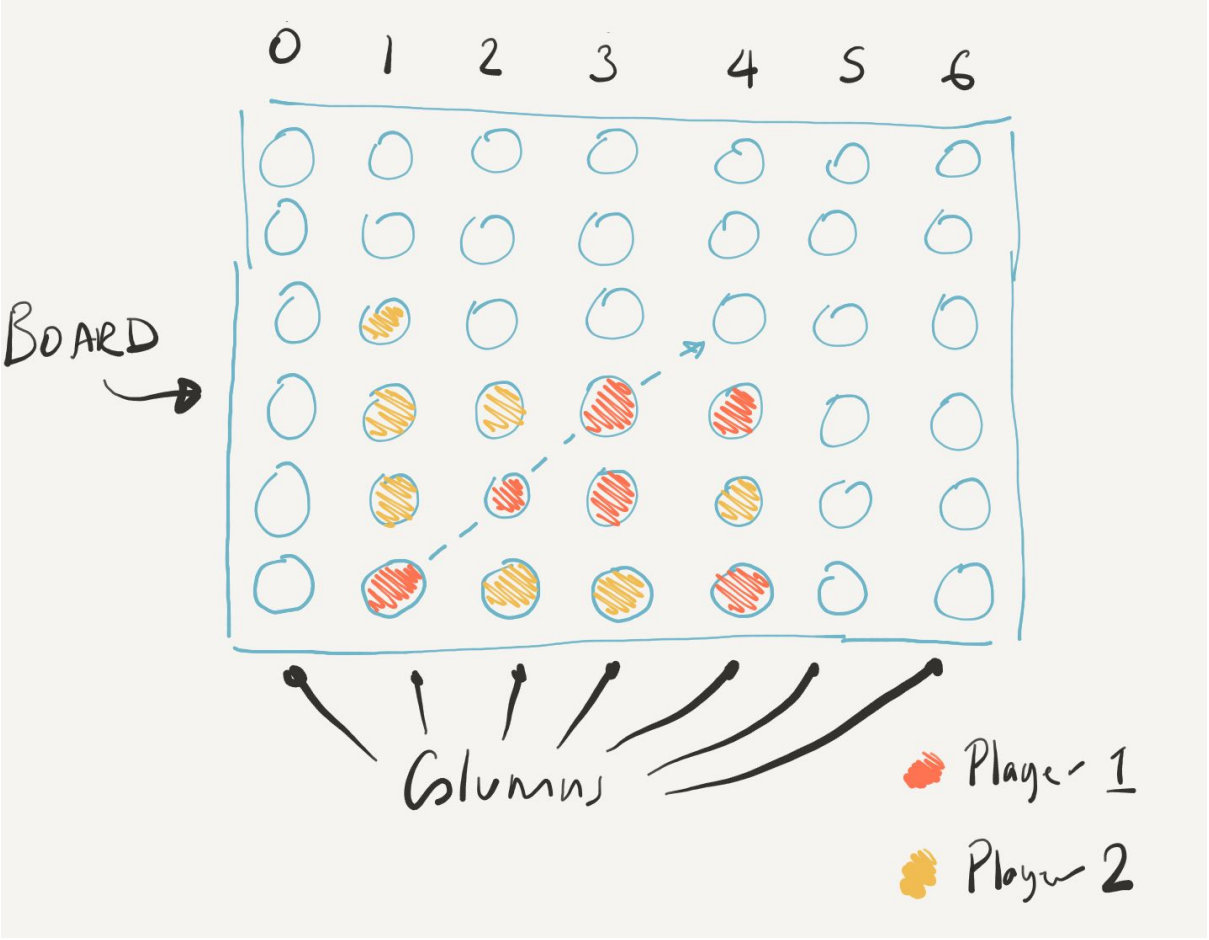


*Figure 1*

In this assessed exercise you will build the objects necessary to create a connect 4 game. The game will be played by two players, each of whom is assigned a character, through the console, with the current state of the board displayed via printing a series of characters like this:

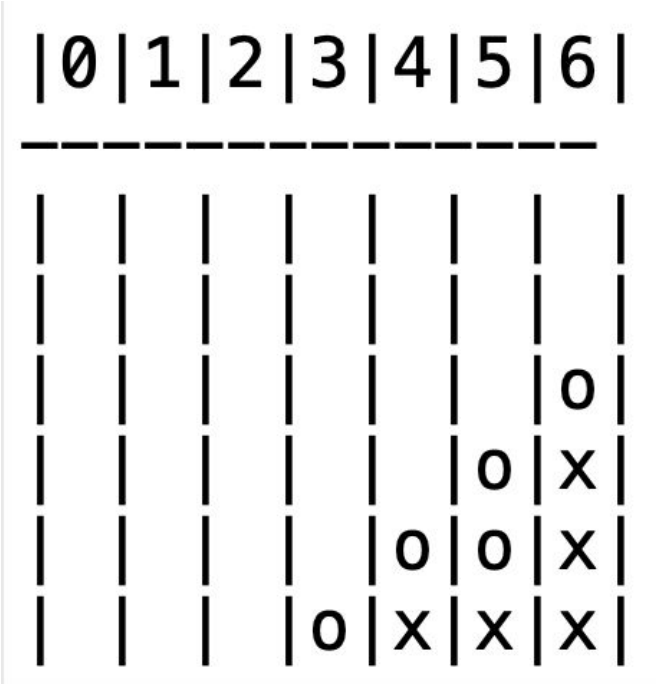


*Figure 2*

## Tasks

As well as the classes listed below, you should create a class that holds a main method (and the static method described in task 4) called **ConnectFour.java**. Ensure your name and matric are given within comments in **every class you submit**. Code should be commented.

1. Players and counters [total: 5 marks]
    a. Create a class to represent a game **player**. It should have an attribute to store the player's **name** (String) and an attribute to store their counter symbol (char), both set via a constructor. [1 mark]
    b. Create a class to represent a game **counter**. It should have only one attribute: a reference to the **player** object that it belongs to, set by a constructor. [1 mark]
    c. Both classes should have getters and toString methods such that the following code would produce the output "Clive, x" [2 marks]:

```
Player p1 = new Player("Clive",'x');
Counter c = new Counter(p1);
System.out.println(c.getPlayer().toString() + ", " +
c.toString());
```

    d. Define a boolean **equals** method in your **Counter** class that returns true if, when it is passed another **Object,** the other object is a **Counter** from the same **Player**. [1 mark]

2. Columns [total: 7 marks]
    a. Create a class to represent a column. It should have a **numRows** attribute that is set by the constructor. It should have an array with **numRows** elements that can store **Counter** objects. Note that the top of the column in the game will be represented by position 0 in your array, and the bottom (i.e. the position the first counter would fall into) will be position **numRows-1**. [2 marks]
    b. Your class should have a boolean method called **isFull** that returns true if the column is full. [1 mark]
    c. Your class should have an **add** method that takes as its arguments a **Counter** object and returns a boolean (true or false). If the column is not full, the **Counter** should be added to the correct position and the method should return true. If the column is full, the method should return false. If you test your **Column** with the following code it should give the output "true, true, true, true, false": [2 marks]

```
Column col = new Column(4);
for(int i=0;i<5;i++) {
      Boolean result = col.add(new Counter(p1));
      System.out.println(result);
}
```

    d. Give your **Column** class a method called **displayRow** that takes a row number as an argument. Assume that the row number is always within an acceptable range. The method should return a String consisting of the counter's character if there is a counter at that position and a String including a space character if there is no counter in that position. [1 mark]
    e. Give your **Column** class a display method that displays each row on a separate line, using the displayRow method. If you test your code with the following example below, it should produce the following output: [1 mark]

```
x
o
x
o
x
o
```

```
Column col = new Column(6);
Player p1 = new Player("Clive",'o');
Player p2 = new Player("Sharon",'x');
for(int i=0;i<3;i++) {
      col.add(new Counter(p1));
      col.add(new Counter(p2));
}
col.display();
```

3. Create a class to represent a **Board**. The class should [total: 5 marks]:
    a. take a number of rows and columns in its constructor (**in that order**). [1 mark]
    b. have a boolean add method that takes a reference to a **Counter** object and a column number as arguments (**in that order**) and returns true if the board successfully adds the counter to that column, and false otherwise. [1 mark]
    c. Have a **toString** method that produces a String representation of the board as shown in Figure 2 (above). [1 mark]
    d. You can test your **Board** class using the following code that should give the output shown in the Figure 2 above. [1 mark]

```
Board board = new Board(6,7);
Player p1 = new Player("Clive",'o');
Player p2 = new Player("Sharon",'x');
board.add(new Counter(p2),6);
board.add(new Counter(p1),3);
board.add(new Counter(p2),4);
board.add(new Counter(p1),4);
board.add(new Counter(p2),5);
board.add(new Counter(p1),5);
board.add(new Counter(p2),6);
board.add(new Counter(p1),5);
board.add(new Counter(p2),6);
board.add(new Counter(p1),6);
```

    e. Add a method to your Board class (**isFull**) that returns true if the board is full and false otherwise. [1 mark]
4. Write a static method (in your **ConnectFour** class) in which two players play randomly (i.e. they randomly choose columns until they find one that isn't full) until the board is full. [total: 2 marks]
5. Winners (this is more advanced and a good pass can be achieved by skipping this part) [total: 3 marks]
    a. Add a method to **Board** that checks if a player has won. Hint: first write a method that checks if, from a particular position, a line (either horizontal, vertical, or diagonal) of Z counters (where Z will typically equal 4) belonging to the same player exists. Then wrap this in a loop over all positions. Feel free to add methods to other classes as necessary. Code should be clearly commented to describe how it works.

# What should you submit?

- A single .zip file containing your .java files. If you're not sure how to make a .zip file, let us know. Please only use .zip and not .rar, .7z, or any other archiving program.
- This should be uploaded to Moodle by the deadline provided on Moodle.