Overview

Connect 4 is a two-player game. Players drop tokens into columns, trying to connect 4 of their tokens either diagonally, horizontally, or vertically. In this project, you'll write logic for a virtual Connect 4 and create an AI opponent using whatever method you prefer.

The project is split into 4 files

- **Main.java**: Handles Swing setup.
- **Connect4.Java**: Yet more Swing, also game logic.
- **Player.java**: Dictates how your bot will "think" and move.

The only one you'll write code in is **Player.java**. Don't modify other classes.

More about Connect 4: https://en.wikipedia.org/wiki/Connect_Four

Steps

Step 0

Download and look over the starter code in **Player.java**.

Step 1

(Player.java)

Choose some tokens to represent the board. You need 3: `MY_TOKEN`, `OPP_TOKEN`, and `EMPTY_TOKEN`.

`MY_TOKEN` is the char to represent the bot.
`OPP_TOKEN` is the char to represent the bot's opponent (in this case, you).
`EMPTY_TOKEN` is the char to represent an empty board space.

E.g.
`MY_TOKEN = '@'`
`OPP_TOKEN = '!'`
`EMPTY_TOKEN = ' '`

But choose whatever's comfortable for you.

Step 2

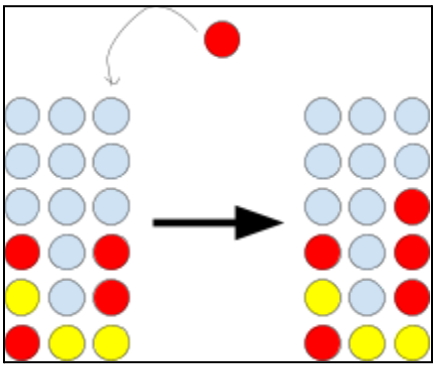
(Player.java)

If you try to run Main.java, you'll probably get an error. We haven't set how the bot moves yet – that's in `public int returnMove()`. This is supposed to return the column to drop your token in.

One simple way to make legal moves is

```
public int returnMove(...){
    return 0;
}
```

Dropping your token on the leftmost column every time. Not a winning strategy, but it's a start. Alternatively, you could make it choose a random column between 0 and COLS.



(What's meant by "dropping" a token)

Step 3
(Main.java)
Try running the program. You should get a playable bot. Not the best bot, but at least it makes moves.

Step 4
(Player.java)
Right now our bot is kind of dumb. It loses pretty easily. We want to make it "smarter."

First of all, we could try to make our bot block obvious wins. Here's how that might work:

```
public int returnMove(){
    // Maybe we should drop in column i?
    // Simulate dropping in column i
    // Check if you won
    // Remove the token you just dropped
    // "I didn't win, maybe try another column" OR "I won, I should drop it there!"

    return ...column number…
}
```

This looks one move in advance and takes wins when it's available. It's a reasonable start.

Step 5

(Player.java)

To do this, you'd need to implement a method that checks if you won. Remember, there are 3 ways to win: horizontally, vertically, or diagonally. E.g.

```
public boolean botWon(){
    for(…each column…){
        // Check horizontal wins
        // Check vertical wins
        // Check diagonal wins
    }

    return ...true if won / false if not…
}

public int returnMove(){
    for(…each column…){
        // Simulate dropping in column i
        if(botWon()){
            // Remove the token you just dropped
            // Yes, let's drop in this column
        } else {
            // Remove the token you just dropped
        }
    }

    return …chosen column…
}
```

Be careful of out-of-bounds errors.

Step 6 [optional]
(Player.java)
If you want, you can try to make the bot look multiple moves ahead. The easiest way to do this is with minimax (https://en.wikipedia.org/wiki/Minimax#Pseudocode). Basically, it brute-forces every possible move, looks even further ahead to see which one is better, and plays the best move given this depth.

The pseudocode in Wikipedia can be repurposed for Player.java. Remember, to find the value of the board, the procedure is the same:

1. Simulate dropping a token
2. Check win/loss/draw
3. Remove token you simulated dropping.

A draw is when the entire board is full (no “empty” chars anywhere).

Warning: This step will likely involve lots of testing and debugging.

Step 7 [optional]
(Player.java)
Some extensions you could possibly take:

1. *Alpha-beta pruning*: currently, the program might take too long if you want a large minimax depth (10, maybe). Alpha-beta pruning can improve this. As always, the Wikipedia article is worth a read.
   https://en.wikipedia.org/wiki/Alpha%E2%80%93beta_pruning

2. *Heuristics*: Any patterns you notice in general? E.g. it tends to be better to play in the center rather than the sides. Combine this with the previous alpha-beta pruning method to find good moves quickly.