PROGRAMMING ASSIGNMENT #1
DOUBLE TROUBLE

FIFTY POINTS

DUE: THURSDAY, MARCH 24, 2022 11 PM

We talked about a game called "Double Trouble", which consists of:

- Three (3) green markers,
- Seven (7) yellow markers, and
- Five (5) orange markers.

Two players take turns removing as many markers of a single color as they wish. The player who removes the last marker wins.

1. "Double Trouble" is really a particular instantiation of THE fundamental combinatorial game. What is the real name of the game, who "solved" it, and when? Where does it show up in popular culture? [2 bonus points]

2. Write a Java program to play "Double Trouble". Your program should contain the input/output functionality necessary to have the user decide who goes first and enter (only!) legal moves in alternation with the computer. The computer need not play *well* but should declare a winner when appropriate and then should terminate correctly. [15 points]

3. Write the necessary Java code that will allow the computer to make a winning move whenever such a move is available to it. You may do this in any way you wish. It can be a stand-alone function or internal to the **main** program. You may use built-in operators or design your own routine. [15 points]

4. Put it all together into a coherent package. The computer should win when it has a winning strategy and should make *random* moves when it does not. Also, the random moves should be independent from one run of the game to another. That is, if you play in such a way that you win, the computer should not lose the same way each time. [20 points]

5. BONUS! Go that extra mile and implement this with something beyond mere text. Develop a GUI! Use colored sticks! I can't think of everything... do something to impress your TA/grader/ME! [5 Bonus points possible]

Also, feel free to personalize your game. Have it play a best-of-$(2n+1)$ tournament, alternating who moves first or randomly deciding who moves first. Have your program celebrate in victory or whimper in defeat. It may not cheat, however. No springs, honest weight!

The overall purpose here is to get you to dig in to Java and get familiar with whatever IDE (Integrated Development Environment) you/we choose to use.

The first part requires basic setup (import, variable definition, etc.), input/output statements, branching (*if*), and looping (*for, while*), perhaps. Stuff you already know how to do.

The second part requires computations, procedures, and functions (perhaps).

The third part requires randomization functions and libraries (perhaps) plus some additional setup to make them work.

Strategy recap: We used the concept of a "Zero Position". The player whose turn it is to move has a winning strategy iff the xor-sum of the position is NOT zero. Any move from a Zero Position must result in a new position with a non-zero xor-sum (no sweeping all the markers to the floor!), and we showed there is always a legal move to reach a Zero Position from any non-zero xor-sum position. Thus, the idea behind a winning strategy is to force a Zero Position on our opponent. That is, we want our opponent to face: $A \oplus B \oplus C = 0$, where A, B, C represent the numbers of markers in our three piles. Here are three forms of a Zero Position:

- $A \oplus B \oplus (A \oplus B) = 0$
- $A \oplus (A \oplus C) \oplus C) = 0$
- $(B \oplus C) \oplus B \oplus C = 0$

Hence,
if $C > A \oplus B$, we remove enough markers from C to create $A \oplus B$, and we win,
OR
if $B > A \oplus C$, we remove enough markers from B to create $A \oplus C$, and we win,
OR
if $A > B \oplus C$, we remove enough markers from A to create $B \oplus C$, and we win.

We can prove that at least one of these is *always* true from any non-zero xor-sum position; therefore, we have a winning strategy from any non-zero xor-sum position.

How does this degenerate with just a single pile of markers, A?
How does this degenerate with just two piles of markers, A and B?
How does this *extend* for n piles of markers, $P_1, P_2, ..., P_n$?