CS 4348 Project 1

The goal: Practice using threads in Java

Due: March 5th, 2024

GitHub Classroom Link: https://classroom.github.com/a/N_s9meaG

The Setup

Your project, including the tests, will come from GitHub. `TestCases` has the tests that you should run to test your code (and will be run by the autograder).

The ***only*** class you need to modify is `ChessMatch`.

The other classes are:

- `ChessMove`: contains a move that will be passed around (you won't create any, but use the ones TestCases creates for you).
- `MoveRecord`: a listing of all the moves made, in order. You will create one of these, share it with your threads, and have them add the moves received from their opponent to this listing.
- `TestCases`: the JUnits for this project. (Note that there is no main() here.) You can run these by pressing the play buttons next to them…easiest to just run the class and get all the tests at once.

The Goal

Use threads to simulate two people playing chess together. Each player will be represented in a thread, and they will repeatedly make a move and then wait for their opponent to make a move. Each thread will record *their opponent's* move as they receive them.

(As you'll be able to tell, these aren't *real* chess moves, but a way to have a bit of fun passing control back and forth between the threads….)

What You'll Do

1. ***Create a private class*** for the first player inside `ChessMatch`. **Name this class with your *first* name** (we're pretending that you're playing yourself). Have it implement `Runnable`.

1. Member variables for this class should be a `List<ChessMove>` and a `MoveRecord`. The constructor should take these types, and assign them to the member variables.
2. This player will go first, so `run()` should look like this
    1. Create a loop for all the moves you got in the constructor.
        1. `synchronize` on `ChessMatch.this`, which is an object both players share. (This ensures that they both lock the same thing.)
        2. Set `playerOneMove` to the next move in your list, then `notify` `ChessMatch.this`.
        3. In a `while` loop, watch for `playerTwoMove` to become non-`null`, calling `wait()` on `ChessMatch.this` until it becomes non-null. (Re-throw any exception as a `RuntimeException`.)
        4. Add player 2's move to the `MoveRecord`, and then `null` it out.
2. **Create a 2nd private class named with your *last* name**. This class is player 2. It is just like player one ***except*** that it plays second, first waiting for the other player's move before sending a move of its own. (In other words, you will `wait` for a move and save it before setting your own move and `notify`ing the shared object.) Use `playerTwoMove` to send your move.
3. In `playMoves()`
    1. Create a new `MoveRecord`.
    2. Create two `Threads`, giving them new instances of the two classes you made. Your **first name** class gets `playerOneMoves` and the `MoveRecord` you created; your **last name** class gets `playerTwoMoves` and the same `MoveRecord`.
    3. `start` the two threads.
    4. Verify that the threads completed. For each thread:
        1. Try to `join(2000)` it—the 2000 ensures that it won't run forever. Turn any exception you get into a `RuntimeException`.
        2. Call `isAlive()` on the thread; if that is `true` (i.e. the thread is still alive), throw a `RuntimeException` to stop the test.

5. Return the `MoveRecord` you made and gave to the two threads; it should now have all the moves recorded in the proper order.