Assignment 1

2-Player Chess

Posted Mon, Feb 5

Due in Canvas on Mon, Feb 26, 11 PM

Late submissions will be accepted, with a 10-point penalty for every 2 hours of lateness past the Feb 26, 11 PM deadline.

Worth 145 points (14.5% of course grade.)

For this assignment you will explore how to apply the object-oriented design ideas you learned in class to design and implement the chess game.
You will work on this assignment in pairs.

We will NOT be using Autolab for assignments. So your submission will be graded once, after the deadline has passed.

Your submission will be AUTOGRADED, so be careful to follow implementation instructions (and constraints) exactly as detailed. There will NOT be any manual inspection of your code, nor will be there any manual inspection of anything your code might print. All credit is solely for structures that are returned from methods that are called on your program by the AUTOGRADER, and checked by the AUTOGRADER against the expect correct structures.

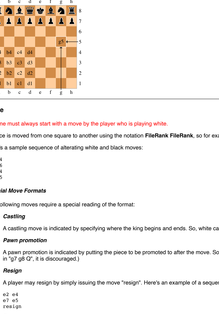
Read the [CCS Academic Integrity Policy for Programming Assignments](#) - you are responsible for this. In particular, note that **"All Violations of the Academic Integrity Policy will be reported by the instructor to the appropriate Dean".**

You will implement the game of Chess for two players. Your program, when launched, should draw the board in text, on the terminal and prompt whomever's turn it is (white or black) for a move. Once the move is executed, the move should be played and the new board drawn, and the other player queried.

Board

The figure below depicts a chess board. Every square on the board is identified in the form of **FileRank** where the File is a column from a thru h, and the Rank is a row from 1 to 8. So, for instance, the square g5 is in column g, row 5.

The white pieces always initially occupy ranks 1 and 2. The black pieces always initially occupy ranks 7 and 8. The queen always starts on the d file.



Move

A game must always start with a move by the player who is playing white.

A piece is moved from one square to another using the notation **FileRank FileRank**, so for example, when the game starts, the move "e2 e4" would move the white pawn from e2 to e4.

Here's a sample sequence of alterating white and black moves:

```
e2 e4
g8 h6
f1 c4
c7 c5
```

Special Move Formats

The following moves require a special reading of the format:

- ***Castling***

  A castling move is indicated by specifying where the king begins and ends. So, white castling king's side would be "e1 g1".

- ***Pawn promotion***

  A pawn promotion is indicated by putting the piece to be promoted to after the move. So, promoting a pawn to a knight might be "g7 g8 N". (Rook is R, bishop is B). If no promotion piece is indicated, it is assumed to be a queen. (Although you can explicitly ask for a queen promotion, as in "g7 g8 Q", it is discouraged.)

- ***Resign***

  A player may resign by simply issuing the move "resign". Here's an example of a sequence of moves that ends with white resigning:

```
e2 e4
e7 e5
resign
```

- ***Draw***

  A player may ask for a draw like this:

```
e2 e4
e7 e5
g1 f3 draw?
```

  Unlike "resign" which is a move all by itself, a draw is asked for by appending "draw?" to an otherwise regular move. Consequently, the move is first carried out, and then the draw is applied.

Illegal Moves

- Every piece must know what moves are allowed on it. If a player attempts an illegal move on a piece, the move should not be executed.
- A piece cannot be moved in a way that would immediately put that player's king under check by the opponent:

```
g2 g3
e7 e5
f1 b5
b8 c6
d2 d3
```

  The white move "d2 d3" would put the white king in check from the black bishop. So it is an illegal move that should not be allowed.

  Note that when a player makes an illegal move, it's still their turn until they make a legal move.

Ending the game

The game may end in one of the following ways:

- ***Checkmate***

  The player that applies the checkmate wins.

- ***Resign***

  A player may resign as described in the previous section. The other player wins.

- ***Draw***

  A player may offer a draw as descried in the previous section.

  For the purpose of this assignment, the other player is **obligated** to accept the draw, even if the player who proposes a draw might be in a losing position.

  Note: There will be no automatic draws (due to unchanging positions over long periods of time, etc).

  You are NOT required to implement termination by threefold repetition, or the fifty-move rule. (You are welcome to include them in your code to make it complete; however, there is no extra credit for either.)

Implementation

You will implement your code in as many classes as you would like, with the following **constraints**:

1. All your classes must be in a package named `chess`. Otherwise we will not be able to grade your program.
2. One of your classes must be the supplied `Chess` class in Chess.java, which is in the `chess` package. (See the line at the top of the file that says `package chess`).

   In `Chess.java`, you will fill in the code for the `start` and `play` methods.

   Our grading script will ONLY call these methods.
3. You will NOT write a `main` method in any of your classes. Instead, you may use the supplied `PlayChess` application in PlayChess.java, which is in the `chess` package.

   This class has a `main` method that you can use to test your implementation.
4. **Do NOT delete or modify any of the original contents of `Chess.java`:**
   - Do NOT change the `ReturnPiece` and `ReturnPlay` classes in ANY way.
   - Do NOT change the `enum Player` in ANY way.
   - Do NOT change the headers of the `play` and `start` methods in ANY way.
5. You may add fields and methods to the `Chess` class as needed.
6. You may also add imports to `Chess.java`, as long as they are from the standard Java SDK (not an external vendor).

Breaking any of these constraints will make your submission ungradable, and will result in a significant penalty on your score.

If you don't know how to make a package named chess and place the given Chess.java and PlayChess.java in it, use ChatGPT or any other online resource to find out. You are responsible for figuring this out and doing it correctly in your development environment.

Development Environment

You may use any Java development environment you are comfortable with, such as Eclipse, NetBeans, IntelliJ, VS Code, etc. Since you will be submitting only source code in the standard Java package structure, what environment you use is not of any consequence to the submission.

However, whatever environment you choose, **you are fully responsible for knowing how to use it** to develop Java applications. This assignment and other assignments in this class may require you to broaden your knowledge in this regard, so you might need to learn how to do certain things yourself by looking up resources online.

We all use different environments in general, so if you run into trouble with yours, there is no guarantee that we can help you because we may have never used your environment to build Java applications.

Returning a result from the `Chess.play` method

The supplied `Chess.java` file contains three classes: the `Chess` class, and two other classes, `ReturnPiece` and `ReturnPlay`. These two classes are specifically set up to return the result of executing the `Chess.play` method.

`ReturnPiece` encapsulates a piece on the board, and `ReturnPlay` encapsulates the collection of all pieces on the board, plus a message that could be null, or could be one of the messages in the `Message` enumeration.

The following table gives examples of all scenarios for different return values when `Chess.play` is executed on a move.

Scenario	Return
Legal move not resulting in a check	`ReturnPlay` instance with pieces on board after move is executed, null message
Legal move resulting in a check	`ReturnPlay` instance with pieces on board after move is executed, message is `CHECK`
Legal move resulting in a checkmate	`ReturnPlay` instance with pieces on board after move is executed, message is `CHECKMATE_WHITE_WINS` or `CHECKMATE_BLACK_WINS`
Illegal move	`ReturnPlay` instance with pieces on board same as previous state of the board (since move is not executed) message is `ILLEGAL_MOVE`
Draw	`ReturnPlay` instance with pieces on board after move is executed (e.g. in "e2 e4 draw?", the move "e2 e4" is executed before draw) message is `DRAW`
Resign	`ReturnPlay` instance with pieces on board same as previous state of the board (since move is not executed) message is `RESIGN_BLACK_WINS` (if white resigned) or `RESIGN_WHITE_WINS` (if black resigns)

Restarting the game from scratch

The `Chess.start` is to be called any time you want to start/restart the game from scratch. This is useful to try a sequence of moves, then reset with `start` for a new sequence from scratch.

IMPORTANT. When we test your submission, the AUTOGRADER will execute several games from scratch. Every time it executes a new game, it will first call your `Chess.start()` method (which should reset the games to the starting start), and then execute moves.

See the sample run in the next section for an illustration.

The `PlayChess` application

You can use this application to test your implementation, by making calls to `Chess.start` and `Chess.play`

Here is a sample run of `PlayChess`:

```
    e2 e4

bR bN bB bQ bK bB bN bR 8
bp bp bp bp bp bp bp bp 7
##    ##    ##    ##  6
##    ##    ##    ##  5
##    ## wp ##    ##  4
##    ##    ##    ##  3
wp wp wp wp    wp wp wp 2
wR wN wB wQ wK wB wN wR 1
  a  b  c  d  e  f  g  h

g8 h6

bR bN bB bQ bK bB    bR 8
bp bp bp bp bp bp bp bp 7
##    ##    ##    bN  6
##    ##    ##    ##  5
##    ## wp ##    ##  4
##    ##    ##    ##  3
wp wp wp wp    wp wp wp 2
wR wN wB wQ wK wB wN wR 1
  a  b  c  d  e  f  g  h

e1 e6

ILLEGAL_MOVE

bR bN bB bQ bK bB    bR 8
bp bp bp bp bp bp bp bp 7
##    ##    ##    bN  6
##    ##    ##    ##  5
##    ## wp ##    ##  4
##    ##    ##    ##  3
wp wp wp wp    wp wp wp 2
wR wN wB wQ wK wB wN wR 1
  a  b  c  d  e  f  g  h

quit
```

Here's another sample run:

```
    e8 e6

ILLEGAL_MOVE

bR bN bB bQ bK bB bN bR 8
bp bp bp bp bp bp bp bp 7
##    ##    ##    ##  6
##    ##    ##    ##  5
##    ##    ##    ##  4
##    ##    ##    ##  3
wp wp wp wp wp wp wp wp 2
wR wN wB wQ wK wB wN wR 1
  a  b  c  d  e  f  g  h

a1 a3

ILLEGAL_MOVE

bR bN bB bQ bK bB bN bR 8
bp bp bp bp bp bp bp bp 7
##    ##    ##    ##  6
##    ##    ##    ##  5
##    ##    ##    ##  4
##    ##    ##    ##  3
wp wp wp wp wp wp wp wp 2
wR wN wB wQ wK wB wN wR 1
  a  b  c  d  e  f  g  h

e2 e4

bR bN bB bQ bK bB bN bR 8
bp bp bp bp bp bp bp bp 7
##    ##    ##    ##  6
##    ##    ##    ##  5
##    ##    ##    ##  4
##    ##    ##    ##  3
wp wp wp wp    wp wp wp 2
wR wN wB wQ wK wB wN wR 1
  a  b  c  d  e  f  g  h

reset (Here your Chess.start method is called to restart from scratch)

e2 e3

bR bN bB bQ bK bB bN bR 8
bp bp bp bp bp bp bp bp 7
##    ##    ##    ##  6
##    ##    ##    ##  5
##    ##    ##    ##  4
##    ##    wp    ##  3
wp wp wp wp    wp wp wp 2
wR wN wB wQ wK wB wN wR 1
  a  b  c  d  e  f  g  h

quit
```

In this second run, the very first move is not allowed because it is supposed to be a white move, which can't move a black pawn. Also, note that when an illegal move is made, it's still the same player's turn until they make a legal move.

The `Chess.start` method is crucial to the AUTOGRADER so make sure you implement it correctly, and test it out to see if it sets up the game to start from scratch as required. This way you (and the AUTOGRADER) can play multiple games in a single run of the program.

Important:

Read the code in `PlayChess` and see how it calls the `Chess.play` and `Chess.start` methods. The AUTOGRADER will make similar calls to both methods.

The `PlayChess` application prints the message and board off the returned `ReturnPlay` object - this is ONLY for your convenience, so you can see a visual of the board for testing.

The AUTOGRADER will NOT print the board. Instead it will compare the structure of the returned `ReturnPlay` object with the expected structure. **If the structures don't match, you will not get credit for the corresponding move.**

Make sure to compile and test your code with `PlayChess`. If you compile your code without including `PlayChess`, your code is not guaranteed to work for the AUTOGRADER and you will lose credit.

You will NOT submit `PlayChess.java`. In case you do, the AUTOGRADER will ignore it.

Grading

- All legitimate basic moves for all pieces
- Castling
- Enpassant
- Promotion
- Identification of check
- Identification of checkmate
- Identification of illegal move
- Resign
- Draw

Note: You are NOT required to implement stalemate.

Submission

Make sure to write your name as well as that of your partner in a comment line at the top of the `Chess.java` file. You don't need to write your names in any other you may have in your application.

You will zip up all the your **source** (.java) files, except `PlayChess.java`, into a submission file named `chess.zip`. (Do not include binary .class files.) Make sure you maintain the directory structure when you zip the files. This is because when you create a package, an equivalent folder is made on the filesystem, and all files in a package go under that folder. Since there is only one package, `chess`, your zip file should have a folder names `chess` that contains all the source Java files.

Make sure to verify that you have zipped correctly by unzipping the `chess.zip` file you made, and confirming that it creates a folder named `chess` under which it places all the contained Java files.

Submit `chess.zip` to Canvas. **Only ONE submission per team, please, so pick which of you will submit! See important points below for penalty in case both of you submit.**

If you don't know how to zip all your Java source files under the package `chess`, use ChatGPT or any other online resource to find out. You are responsible for figuring this out and doing it correctly in your development environment.

Important:

- ONLY submissions on Canvas will be graded. If you make multiple submissions, the latest submission will override the previous, so only the last submission will be graded.
  We will ignore any code that is emailed to us.
  We will not inspect or accept code that is on your computer, period.
- If your program does not compile because of incorrect package structure, and we need to restructure it to make it compile, you will incur a penalty on your score.
- If there are two submissions for your team, you will be penalized 10 points on your score, and we will only grade the later submission. In particular, if one of you submitted by the regular deadline, and the other submitted late, the late submission will be graded (see point below) with penalty for lateness on top of the 10 point penalty for submissions from both members.
- Anything submitted after Feb 26, 11 pm will be treated as a late submission. Remember, there is a 10 point penalty for every 2 hours of lateness past the deadline. For example, if your late submission is on Feb 27 at 4:55 am, you will lose 30 points: 10 pts (11pm-1am) + 10 points (1am-3am) + 10 points (3am-5am). Once your lateness slides into the next 2-hour window, you will be penalized the entire 10 points, even if it is in that window by seconds. No exceptions.

Frequently Asked Questions

Q: Can we assume that all moves sent as parameter to the `Chess.play` method will be syntactically correct?
A: Yes. All inputs will have legit file and rank, will have exactly one space between components, such as "e2 e4", or "g7 g7 N". There may be leading or trailing spaces around the whole move, so you will need to handle that.

Q:Do we have to implement a rule where a player is not allowed to make an otherwise valid move if it will immediately put their king in check?
A: Yes. This qualifies as an illegal move.

Q:We saw rules elsewhere on castling/dead position/what-have-you that go beyond the description on the linked Wikipedia page. Should we implement those?
A: No, we are going with the Wikipedia page only. We are not all practiced Chess players.

Q:Will I get any credit if the `ReturnPlay` object returned by `Chess.play` is incorrect, but I can show that the board is printed correctly?
A:No, you will not get any credit. The AUTOGRADER checks the value of the `ReturnPlay` object on each move, and if it is incorrect, you will not get any credit. There is no manual inspection of anything that may be printed by your program.

Q:I coded a bunch of print statements in my program for testing, but forgot to delete them before submitting. Will this mess up the grading?
A:No, it will not mess up the grading. Anything printed by your program will be ignored.

Q:Can I add code to the `PlayChess.java` file?
A:Yes, you may. Since you will not be submitting `PlayChess.java`, you may add code to it to help with your testing. But make sure you don't change the way it calls `Chess.play` and `Chess.start` since the calls are written to exactly match the requirements of the method headers for `play` and `start`.

Q:If the user is asks for a draw but the move they entered is illegal, is it a draw or an illegal move?
A:Since the move happens before the draw, the move is executed first. So it is an illegal move, that is what would be reported, not draw. Which means the player will get another chance to make a move.

Q: If the user enters draw (paired with a legal move), or resigns, or checkmates, do we have to stop taking input or end the game?
A: Since you are not writing an application with a main method, it is up to the caller of the play method (for example PlayChess.java) to terminate the program. From your code, you will simply return the required result from the play method when draw/resign/checkmate happens, and let the caller do the rest. (For instance, the caller may stop the program, or may call your start method to start over from scratch.