CCCS 300 Assignment 3

Introduction

This assignment is aimed to help you practice using arrays, classes, and objects. It also reinforces earlier concepts such as methods and loops.

This assignment is challenging, and you may feel that there is not a straightforward connection between the lecture and the assignment. This feeling is normal. If you get a job as a programmer, you will often be faced with problems that you don't quite know how to solve at first. Programming is really about persistence and overcoming obstacles. You will have to think very critically about how to use the material covered to complete this assignment. It won't be easy

This assignment is out of 100, and accounts for 15% of the final grade.

More suggestions and guidelines are given at the end.

Due Date

November 06, 2019

What to submit

Submit only the following two files onto MyCourses:

- Board.java
- Knight.java

Late Policy

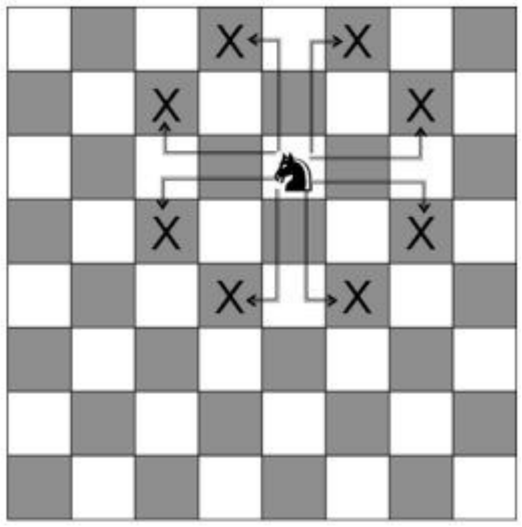
- If you submit late, you run the risk of having your submission rejected, in which case you will receive zero for this assignment.

Background

This assignment is inspired by chess, and to do this assignment you need to have a basic understanding of chess and how the knight moves.

Chess is a two-player turn-based board game. One player has white pieces and one player has black pieces. The two players take turns making moves, and during each move a player must move one of his own pieces. You can read more about the game on Wikipedia if you've never played chess before (https://en.wikipedia.org/wiki/Chess).

The knight is a piece that has an L-shaped move set. In the following figure, we see that a knight can move to 8 possible locations, relative to where it is currently.



Overview

In this assignment, you will implement TextChess, which allows the user to play an adapted chess game on the command line. Below we show example output of TextChess. User input is in red.

```
Generating chess game of size 12 by 6 with 8 knights

 A B C D E F G H I J K L
-------------------------
| | | | | | | | | | | | | 1
-------------------------
| | | | | |N|N|N| |n| | | 2
-------------------------
| | | |n| |N|n| | | |N|N| 3
-------------------------
| | | | | | | | | | | | | 4
-------------------------
|n| | | | | | | | | | |n| 5
-------------------------
|n| | |n| |n|N| | |N| | | 6
-------------------------
Player (upper case) move: F 2 D 3
AI moves from J 2 to L 3 and captured your piece.
Player lost 1 piece(s).
AI lost 1 piece(s)

 A B C D E F G H I J K L
-------------------------
| | | | | | | | | | | | | 1
-------------------------
| | | | | | |N|N| | | | | 2
-------------------------
| | | |N| |N|n| | | |N|n| 3
-------------------------
| | | | | | | | | | | | | 4
-------------------------
|n| | | | | | | | | | |n| 5
-------------------------
|n| | |n| |n|N| | |N| | | 6
-------------------------
Player (upper case) move: J 6 L 5
AI moves from L 3 to K 1
Player lost 1 piece(s).
AI lost 2 piece(s)

 A B C D E F G H I J K L
-------------------------
| | | | | | | | | | |n| | 1
-------------------------
| | | | | | |N|N| | | | | 2
-------------------------
| | | |N| |N|n| | | |N| | 3
-------------------------
| | | | | | | | | | | | | 4
-------------------------
|n| | | | | | | | | | |N| 5
-------------------------
|n| | |n| |n|N| | | | | | 6
-------------------------
Player (upper case) move:
```

To run TextChess, the user issues a command in the following format

```
java textchess.TextChess 12 6 8 4
```

The first two arguments 12 and 6 indicate the size of the board. Here, we generate a 12 by 6 board that is 12 columns wide and 6 rows tall. The third argument 8 indicates how many pieces each player gets. To keep things simple, the knight is the only piece that is available in this game. So, in this example, each player gets 8 knights, for a total of 16 pieces on the board. These 16 knights are randomly placed on the board. The last argument 4 is a random number seed, which the program will use to generate random numbers.

Normally in chess there are black and white pieces. However, there is no easy way to have black and white on the command line. So, white knights are denoted by an uppercase N, and black knights represented by a lowercase n. At this point, the output on the previous page should start to make sense.

In this game, the player will always use white pieces (uppercase N), and the player will play against an AI, which will use the black pieces (lowercase n)

At the start of the game, your program will generate a new game with the specified board dimensions and the specified number of knights randomly placed on the board. It will then print the board to the command line and wait for the user to input a move. Using the coordinates laid out on the top and to the right of the board, the user specifies a move. The move `F 2 D 3` moves the player's white knight from location F 2 to D 3. This move captures the AI's piece at D 3.

After the player moves, the AI will move and will output something like

```
AI moves from J 2 to L 3 and captured your piece.
```

The AI will let you know if it took one of your pieces.

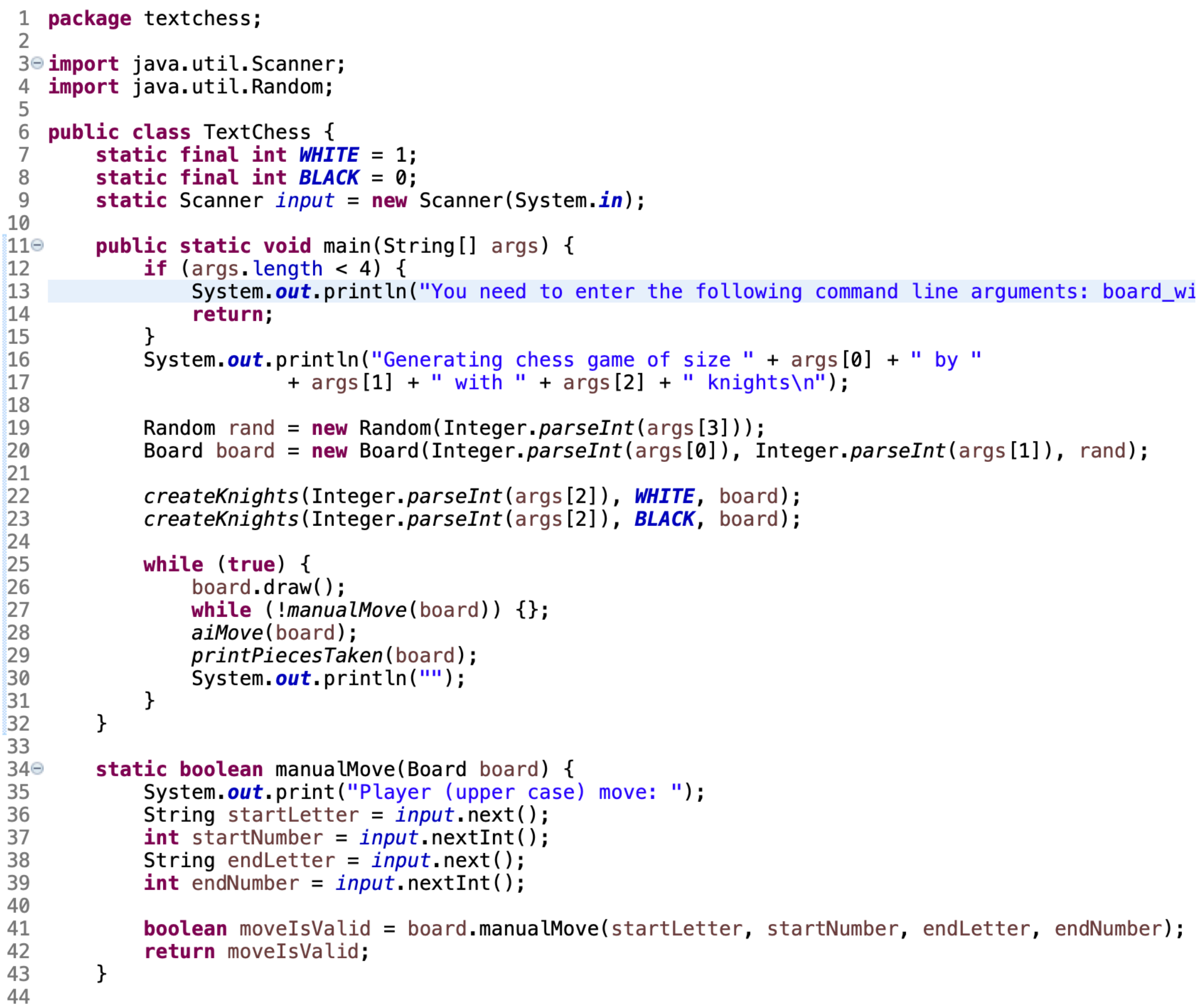
Next, the program prints out the number of pieces each side as lost

```
Player lost 1 piece(s).
AI lost 2 piece(s)
```

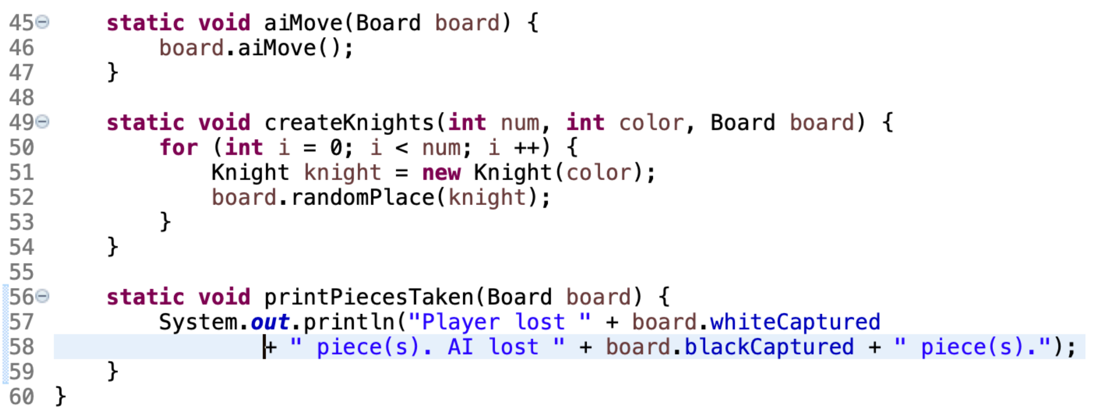
Finally, the program prints out the updated chess board, and waits for the user to make another move.

Program Specifications

In this assignment, you are given the following file named <u>TextChess.java</u> as a starting point.



```java
package textchess;

import java.util.Scanner;
import java.util.Random;

public class TextChess {
    static final int WHITE = 1;
    static final int BLACK = 0;
    static Scanner input = new Scanner(System.in);

    public static void main(String[] args) {
        if (args.length < 4) {
            System.out.println("You need to enter the following command line arguments: board_wi
            return;
        }
        System.out.println("Generating chess game of size " + args[0] + " by "
                + args[1] + " with " + args[2] + " knights\n");

        Random rand = new Random(Integer.parseInt(args[3]));
        Board board = new Board(Integer.parseInt(args[0]), Integer.parseInt(args[1]), rand);

        createKnights(Integer.parseInt(args[2]), WHITE, board);
        createKnights(Integer.parseInt(args[2]), BLACK, board);

        while (true) {
            board.draw();
            while (!manualMove(board)) {};
            aiMove(board);
            printPiecesTaken(board);
            System.out.println("");
        }
    }

    static boolean manualMove(Board board) {
        System.out.print("Player (upper case) move: ");
        String startLetter = input.next();
        int startNumber = input.nextInt();
        String endLetter = input.next();
        int endNumber = input.nextInt();

        boolean moveIsValid = board.manualMove(startLetter, startNumber, endLetter, endNumber);
        return moveIsValid;
    }
```



```java
    static void aiMove(Board board) {
        board.aiMove();
    }

    static void createKnights(int num, int color, Board board) {
        for (int i = 0; i < num; i ++) {
            Knight knight = new Knight(color);
            board.randomPlace(knight);
        }
    }

    static void printPiecesTaken(Board board) {
        System.out.println("Player lost " + board.whiteCaptured
                + " piece(s). AI lost " + board.blackCaptured + " piece(s).");
    }
}
```

You will notice that this file is in a package called textchess, so you will need to set up your project accordingly by creating the textchess package. You will also see that this code makes use of a Board class and a Knight class. **Your job is to complete the Board class and the Knight class, which are written**

inside Board.java and Knight.java respectively. When running TextChess, the user will invoke the main function of TextChess, which in turns calls upon the Board and Knight classes you will create.

The programs you submit must compile. I must be able to compile TextChess.java after adding your Board.java and Knight.java to my project. If I cannot compile all three java files, then you will receive ZERO.

0) Restrictions

For this assignment, you are free to write whatever code you feel is needed, with the following exceptions

- All code you write must be contained inside Board.java and Knight.java, with the Board class defined in Board.java, and the Knight class defined inside Knight.java
- For the **final version** of your program, you **MUST NOT** change the provided TextChess.java file. I understand that when you first start out, you might want to change the file for testing or debugging purposes. However, note that when you submit your assignment, you do not submit TextChess.java, which means we will assume that you have not changed it
- Your Board.java and Knight.java **MUST NOT** have a main function
- You **MUST NOT** instantiate a Random object anywhere in Board.java or Knight.java. On line 16 of TextChess.java, you will notice that when we create a Board object, we pass in a Random object. To generate random numbers, you **MUST USE** this random object. **DO NOT USE** Math.random anywhere in your program.

If you do not follow these instructions, the grader will deduct marks as he sees fit.

1) Randomly placing knights (20 marks)

On line 22 and 23, we call the createKnights function to place white and black knights onto the board. The createKnights function uses a loop to create the necessary number of knights. Lines 51 and 52 are the key lines. Line 51 creates a knight object of the specified color, and 52 places that knight randomly on the board. You will need to implement the randomPlace method inside your Board.java, that takes a Knight object, and places it randomly on the board. You will need to use the Random object that is passed into the constructor of the Board class.

The main challenge here is that you cannot place a knight on a square that already has another piece. You will have to think creatively about how to solve this problem. There is not a single right answer.

2) Drawing the board (5 marks)

The while loop starting on line 25 is the main loop of this program, which will repeatedly draw the board and ask the user to make a move. Line 26 invokes the draw method of the Board class. This should draw the board, and the chess pieces on the board. This is task is partially completed already – the program is already able to draw the grid and the axis labels. However, you need to modify the code in the Board class to correctly show the Knight pieces on the board. For simplicity, we will assume the width of the board does not exceed 26.

3) Player move (30 marks)

Line 27 calls the manualMove method repeatedly until the user enters a valid move. You need to implement this method. The manualMove method takes as input the starting and ending coordinates of a move. This method must first check if the move is valid. A move is **not valid** under the following circumstances

a) The start position does not contain a piece that the player controls
b) The end position is outside the board
c) The end position contains one of the player's own pieces
d) The specified move is not in the knight's L-shaped move set

If any of the above cases is true, then your manualMove method should simply returns false without moving any pieces. It must also output what the problem is.

- For case a), your program must output "The player does not have a piece at X X".
- For case b) your program must output "End position is not inside the board".
- For case c) and d), your program must output "Invalid move".

See below for example output

```
Generating chess game of size 12 by 6 with 8 knights

 A B C D E F G H I J K L
-------------------------
| | | | | | | | | | | | | 1
-------------------------
| | | | | |N|N|N| |n| | | 2
-------------------------
| | | |n| |N|n| | |N|N| 3
-------------------------
| | | | | | | | | | | | | 4
-------------------------
|n| | | | | | | | | | |n| 5
-------------------------
|n| | |n| |n|N| | |N| | | 6
-------------------------
Player (upper case) move: A 1 A 2
The player does not have a piece at A 1
Player (upper case) move: F 3 H 2
Invalid move
Player (upper case) move: H 3 M 1
End position is not inside the board
Player (upper case) move: F 2 F 1
Invalid move
Player (upper case) move: L 5 J 4
The player does not have a piece at L 5
Player (upper case) move:
```

If the move is valid, your method must move that piece and update the board accordingly.

Your program must properly handle the capturing of enemy pieces. If the player moves his/her knight (uppercase) onto one of the AI's knights (lowercase), then the AI's knight is removed from the board. In a chess game, the more pieces you capture the better.

4) AI move (15 marks)

Line 28 calls aiMove, which in turn calls the aiMove method of your Board object on line 46, which you must implement. The AI may move any black (lowercase) piece, but the move **must be valid**. This is a fairly dumb AI because it will not try to take the player's pieces. However, you can receive marks for this part just by implementing an AI that can follow the rules.

After moving the AI's piece, you must print the following:

- Indicate which piece was moved by printing out the starting and end location of that piece.
- You must also output whether the AI captured one of the player's pieces.

Please refer to the example at the beginning of Overview for example output.

5) Show pieces taken (5 marks)

On line 29 we call the printPiecesTaken method, which prints out the number of pieces that have been taken for each player. This method relies on the whiteCaptured and blackCaptured properties of your Board class, which you need to ensure is properly updated when pieces are taken.

6) A smarter AI (25 marks)

To get full marks on this assignment, your AI must become smarter. A smart AI must have the following behavior:

- If the AI can take one of the player's pieces this turn, it must take it
- If the AI cannot take one of the player's pieces this turn, then it must make a valid move

7) Winning condition

You might be wondering how someone wins this game. For simplicity, we will not be implementing a winning condition. You can think of the objective of the game as taking as many of the opponent's knights as possible.

Additional Suggestions

Due to the difficulty of this assignment, the grading scheme is set up so that you can obtain partial marks. For example, if you just complete parts 1) 2) 3) correctly, then you would receive at least a passing grade for this assignment. If you show progress towards the other sections, you will be able to gain partial marks for the other parts.

I suggest that you focus on 1) 2) 3) first to keep things manageable. Note that you can implement a fully playable game even if the AI never moves. After finishing part 3) you will then realize that the validity checking of 3) can be used for parts 4) and 6) as well.