American University of Armenia, CSE
CS120 Intro to OOP A, B, C
Spring 2022

Homework Assignment 7

Due Date: Saturday, March 26 by 23:59 electronically on moodle

During this and the next few homework assignments, you will be developing a chess application by gradually adding more features and re-implementing and improving existing code.

Please consult the wikipedia article for chess and make sure you are familiar with the rules and the terminology. Moreover, you can play chess with AI or human players from around the world at `lichess.org`. This platform is open-source and does not require registration.

You are allowed to write additional methods if you need them during development, but you must not change the headings of any of the specified or existing methods.

Your use of components from other packages is limited to `Scanner` from `java.util` and wrapper classes. Your code should follow good OOP practices and directly apply the principles of abstraction and encapsulation.

1. **(20 points)** Start by creating a class named `Position` to represent a position on the chessboard. It should include two instance variables of type `int` to represent the rank and the file of the position.In this internal representation, these will be integer coordinates where the position “A8” is equivalent to $(0, 0)$, “A1” to $(7, 0)$, “H1” to $(7, 7)$, etc.

   An alternative would be to represent a position with a string like “B5” directly. But a string representation is not convenient in use. Instead, this class communicates positions in the form of pairs of coordinates in the board.

   Include in the class

   - a no-arg constructor to initialise the position of the top-left corner of the chessboard, i.e. $(0, 0)$;
   - a copy constructor;
   - a constructor that receives two `int` parameters and initialises both instance variables accordingly;
   - an accessor and a mutator for each instance variable (you may want to introduce sanity checks inside the mutators);
   - a method with the heading
     `public String toString()`
     that generates and returns a string representation of the position with a format like “C2”.
   - a static `generateFromString` method;
   - a static `generateFromRankAndFile` method;
   - a static `appendPositionToArray` method.

   The `generateFromString` and `generateFromRankAndFile` methods deal with the same task—to generate a new `Position` object from a single string representation argument like "B3" or a pair of integer arguments for coordinates. If the arguments supplied do not comply with the expected format and value ranges, the methods should just return `null` instead of a real object.

Finally, the heading of the last method should be
`public static Position[] appendPositionToArray(Position[] arr, Position p)`.
This method is going to prove a useful utility in the future. As the name suggests, its purpose is to append a given position `p` to the end of an array of positions `arr`. Since Java doesn't allow resizing arrays, this is achieved by actually creating a new larger array and putting the positions in it (a shallow approach is OK).

Think carefully about the access levels you choose for the different class members. Try to avoid unjustified visibility of members, particularly such that the creation of objects containing invalid values is prevented.

2. **(10 points)** Introduce a second class named `Move` to represent a move that a player makes during the game. This class should contain two instance variables to denote the origin and destination positions of the move. Include in the class

   - a constructor that receives two `Position` parameters and initialises both instance variables accordingly;
   - a copy constructor (think carefully about how deep the copy needs to be);
   - an accessor for each instance variable (Beware of privacy leaks!);
   - a method with the heading
     `public String toString()`
     that generates and returns a string representation of the move with a format like "C2 C4".

3. **(25 points)** The class named `Chess` is an important one. This class should store the state of an ongoing game of chess and provide means for changing that state. The state of the game includes

   - the board and the pieces on it as an instance variable `board` of type `char[][]` and
   - a counter for the current number of moves in the game.

   The matrix `board` represents chess pieces as characters 'K', 'Q', 'R', 'B', 'N', 'P': uppercase for whites and lowercase for blacks. You should also choose another special character to denote empty positions on the board.

   Board dimensions and the empty character should also be declared here as class, i.e. static, constants.

   Include in your class a no-arg constructor. It should initialise all instance variables to default values: no moves yet an empty board (for now).

   Add into your class the following methods:

   - an accessor method that generates and returns a *deep* copy of the board (Why should one use a copy here?);
   - an accessor `getTurn` that, using the current number of moves, returns either `0` or `1` to indicate white's or black's turn, respectively;
   - a method that checks if the game is over or if it still continues (it should just return `false` in the scope of the current assignment, but will be properly implemented in future homework assignments);

- a method `isWhitePieceAt` that, given a position, checks if that position on the board contains a white piece;
- a method `isEmpty` that, given a position, checks if that position on the board is empty;
- an accessor `getPieceAt` method that, given a position, returns the character representing the piece at that position on the board;
- a `reachableFrom` method;
- a `performMove` method.

Different types of chess pieces move according to different patterns. These set of rules determine which squares a piece can move to, given the position it is currently at. For every chess piece at any given time, there is a set of reachable squares. While that set might be empty for certain pieces in certain situations, generally speaking, there are no immovable pieces.

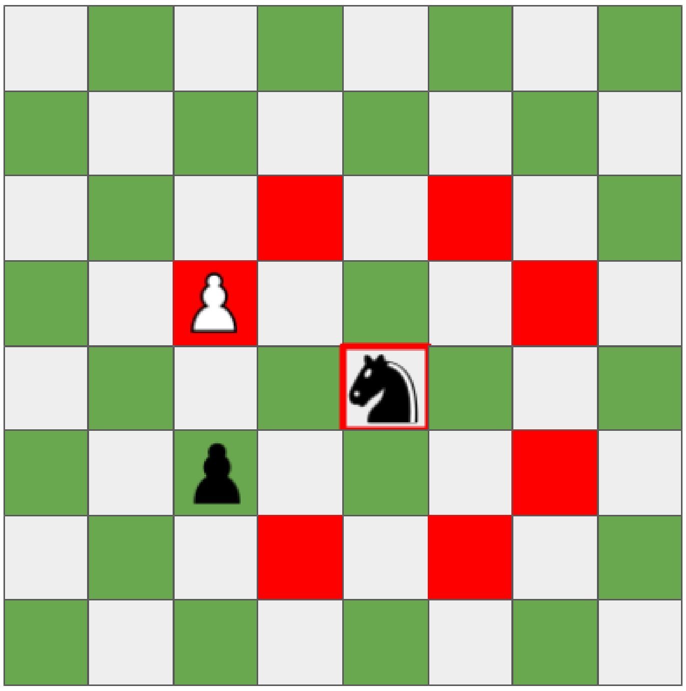
A square is deemed reachable if it falls within the movement pattern of the piece, is not occupied by a friendly piece of the same color and the path is not blocked by a piece of any color. Knights are an exception to the path condition and can "jump over" other pieces.

The method `reachableFrom` should have a heading
`public Position[] reachableFrom(Position origin)`
and should identify the set of all positions that can serve as destinations for a move from the given origin. If the origin is not a real object or is empty, the method should just return `null`. Otherwise, it should return a listing of all eligible destinations based on the type of piece in origin. In the current homework assignment, we only consider knight pieces. Thus, your implementation should use corresponding functionality from the class `Knight`, as described in the task below.

Finally, the `performMove` method, given a `Move` object, should attempt to make the specified move. If the move is valid, i.e. the destination is in the set reachable from the origin, the method should correctly update the contents of `board`, increment the number of moves, and return `true`. Otherwise, the method should return `false`.

Think carefully about which, if any, of these methods don't rely on object-specific data and make them `static`.

4. **(15 points)** As mentioned above, in this assignment we only consider one type of chess pieces—knights. Create a class `Knight` to contain only a single static method `reachablePositions` with the heading
`public static Position[] reachablePositions(Chess chess, Position p)`. As you may guess, the method should generate and return a set of all the positions into which a knight might perform a valid move from the given position `p` in the ongoing `chess` game. Note how this method implementation will most probably use quite a few of the methods from the already defined classes.



5. (20 points) The class `ChessConsole` is responsible for providing a console user interface for your program. First of all, it needs to keep track of the ongoing game, i.e. there should be an instance variable `game` of type `Chess`.

Next, it should contain the following method:

```java
public void play() {
    Scanner sc = new Scanner(System.in);
    String inputLine;
    game = new Chess();

    print();

    while (!game.isGameOver()) {
        if (game.getTurn() == 0) {
            System.out.println("White's move: ");
        } else {
            System.out.println("Black's move: ");
        }

        inputLine = sc.nextLine();
        String[] input = inputLine.split(" ");
        if (input.length == 1) {
            print(Position.generateFromString(input[0]));
        }
        else if (input.length == 2) {
            Move m = new Move(Position.generateFromString(input[0]),
                              Position.generateFromString(input[1]));
            boolean success = game.performMove(m);
            if (!success) {
                System.out.println("Invalid move. Please try again.");
            }
            print();
        }
    }
}
```

Make sure you clearly understand how the method works and what it does.

Add a method `print` to print the board and the pieces on it.

Consult the guide regarding escape sequences for console output if you would like to use colors in your method.

Consult the unicode table for chess pieces if you wish to use those characters instead of the small and capital letters defined in our model. It should be noted that this only applies to printed outputs and not the underlying model.

Finally, overload the `print` method such that it highlights a selected piece, given as a `Position` parameter, along with the squares to which it can move to.

Make sure to reduce code repetition as much as possible between the two `print` methods.

6. **(5 points)** Complete your code with a `Main` class to include a `main` method that creates a `ChessConsole` object and invokes the `play` method on it.

   Modify the no-arg constructor of the class `Chess` to test your current code with different initial boards containing one or more knights.

7. **(5 points)** Complete your code with proper documentation comments and use `javadoc` to generate corresponding API specification pages.

   Finally, in 2–3 sentences explain which of your classes are mutable and which are not.