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

Homework Assignment 8

Due Date: Saturday, April 9 by 23:59 electronically on moodle

This homework assignment builds upon the previous one. You are welcome to use your solution to HW7 as a basis for HW8 or, alternatively, the HW7 model solution accompanying this assignment on Moodle. Note in either case the addition of the class `Rook` that represents rook pieces (briefly discussed below).

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, encapsulation, inheritance and polymorphism.

1. **(4 points)** First, you should introduce packages into your code by moving the different classes into a few packages:

   - `Main` into package `am.aua.chess`;
   - `Position`, `Move`, `Chess`, `Knight` into package `am.aua.chess.core`;
   - `ChessConsole` into package `am.aua.chess.cli`.

   Package `am.aua.chess.core` is responsible for grouping functionality related to the game and its rules. Package `am.aua.chess.cli` supports a command-line interface for playing the game. Finally, package `am.aua.chess` contains the main entry point to the game.

   Any further class in this assignment will be added into one of these three packages. Think carefully about where you place each new class.

   Remember that this implies placing the source files into corresponding directories (folders) and also adding package statements at the top of those source files. Furthermore, now that the different classes are in different packages, you may need a few import statements to make them "see" each other.

2. **(3 points)** Inside class `Chess`, add an enumerated type called `PieceColor`. It should provide two constants—`WHITE` and `BLACK`—to represent the colors of the pieces belonging to the two players.

   Inside `Position`, modify `appendPositionToArray` method into `appendPositionsToArray` to allow the addition of multiple positions (rather than just one) to the end of an array by using a *vararg specification*.

3. **(9 points)** A major change in this assignment is the introduction of a type to represent the notion of a chess piece. Introduce a class `Piece` for this purpose. It should contain only one instance variable denoting the color of the piece.

   There should be two constructors: a no-arg constructor making the piece color `WHITE` by default, and a constructor that initializes the piece color from an argument.

   Provide an accessor for the piece color.

In addition, provide a method with heading
`public Position[] allDestinations(Chess chess, Position p)`
that just returns `null`. Later, this method will be *overridden* in the subclasses of `Piece`. It is intended to return a set of all positions into which a piece might perform a valid move from the given position `p` in the ongoing `chess` game.

4. **(7 points)** As a result of introducing `Piece`, you need to modify `Knight` to become its subclass. Introduce two constructors (no-arg and one with a `Chess.PieceColor` parameter) and make them rely on parent constructors.

   Refactor the old static `reachablePositions` method into an overridden `allDestinations` method by keeping the method body (and only adapting it to API changes) but replacing the method heading.

   Add an overridden `toString` method that generates the string `"N"` for a white knight and `"n"` for a black knight.

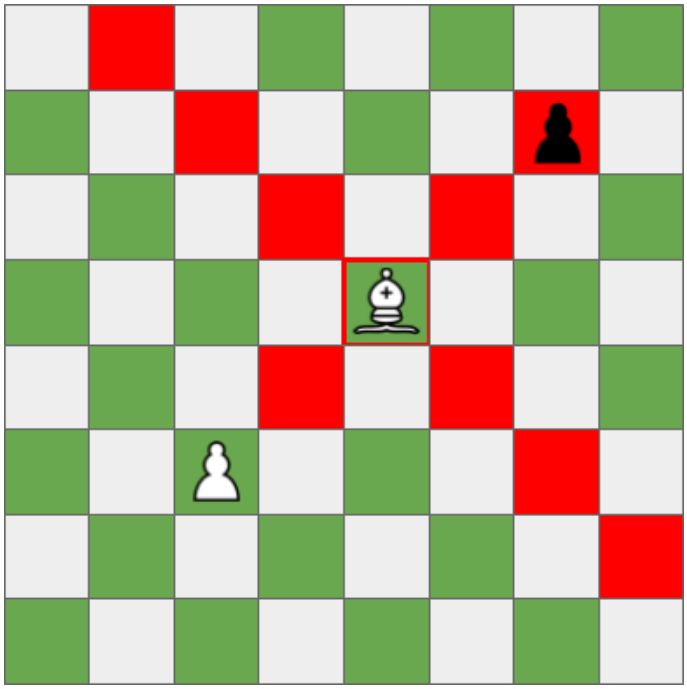
   Place the new `Rook` class supplied with this assignment into the same package as `Knight`. Make sure you understand all the details of its implementation.

   One important aspect that makes `Rook` different from `Knight`, is that it maintains a boolean flag to track whether this rook has ever moved in the course of the game. Later, if you introduce castling*, you will be able to rely on this flag to check applicability of castling.*

   Another important aspect is the separation of the method `allDestinations` and the static method `reachablePositions`. The latter is going to prove useful when introducing queen pieces.

5. **(9 points)** Create a class `Bishop` to represent bishop pieces. Feel free to use code patterns from `Knight` and `Rook` when completing this class. It should contain:

   - a no-arg constructor;
   - a constructor with a single `Chess.PieceColor` parameter;
   - `toString` method;
   - `allDestinations` method;
   - static `reachablePositions` method.



6. (9 points) Create a class `Queen` to represent queen pieces. It should contain:

- a no-arg constructor;
- a constructor with a single `Chess.PieceColor` parameter;
- `toString` method;
- `allDestinations` method.

Hint: The static `reachablePositions` methods from `Rook` and `Bishop` can be used inside `Queen`'s `allDestinations` method to model a queen piece's applicable moves together.

7. (14 points) Create a class `Pawn` to represent pawn pieces. It should contain:

- a no-arg constructor;
- a constructor with a single `Chess.PieceColor` parameter;
- `toString` method;
- `allDestinations` method.

Remember that pawns have a choice of a single or a double forward-step as their first move into an empty square and only a single forward-step (into an empty square) afterwards. In addition, they capture opponent's pieces with a single forward-step diagonally.

8. (12 points) Create a class `King` to represent king pieces. It should contain:

- a boolean `hasMoved` flag;
- a no-arg constructor;
- a constructor with a single `Chess.PieceColor` parameter;
- a third constructor with a `Chess.PieceColor` and a `boolean` parameters;
- `toString` method;
- `allDestinations` method.

9. (24 points) Now, having introduced all the different pieces, the `Chess` class should be refactored.

(a) Get rid of the empty character constant and the `isWhitePieceAt` method.

(b) Change the type of the instance variable `board` to become a matrix of `Pieces` instead of `chars`.

(c) Introduce another constructor that, given a `String` and a `PieceColor`, sets up an ongoing chess game.

The string parameter represents the positioning of the different pieces on the chess board. Its length should be 64 to represent the squares of the board in a row-major order, according to this table for mapping pieces:

'R'/'r'	white/black rook that hasn't moved yet
'S'/'s'	white/black rook that has moved already
'N'/'n'	white/black knight
'B'/'b'	white/black bishop
'K'/'k'	white/black king that hasn't moved yet
'L'/'l'	white/black king that has moved already
'Q'/'q'	white/black queen
'P'/'p'	white/black pawn

The second parameter indicates which player is about to make a move.

This constructor should ensure that there is *exactly* one black and *exactly* one white king present on the board. Otherwise, it should abort the program with meaningful messages printed to the user.

(d) Modify the no-arg constructor to fill the board with the standard initial configuration of a chess board.

(e) Update `getBoard`, `isEmpty`, `getPieceAt` methods to reflect the type change in the variable `board`.

(f) Update `getTurn` to use the enumerated type.

(g) Update `reachableFrom` method to make use of the `allDestinations` method from `Piece`. Note how `reachableFrom` can be defined in just a few statements if you rely on *polymorphism*.

You may want to introduce static constants here to help with the implementation of the `Pawn` class.

10. (4 points) Inside `ChessConsole`'s `play` method reflect the changes to the `Chess` API. In addition, add logic for ensuring correct turn-taking. That is, if a position is specified by the user (for highlight or as a move origin), but that position contains an opponent's piece, the program should warn the user of an invalid request.

For testing purposes, inside the `play` method, try creating custom board configurations instead of the starting configuration by changing the `Chess` constructor call.

11. (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.