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

Homework Assignment 10

Due Date: Friday, May 6 by 23:59 electronically on moodle

This homework assignment builds upon the previous three. You are welcome to use your solution to HW9 as a basis for HW10 or, alternatively, the HW9 model solution accompanying this assignment on Moodle.

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 code should follow good OOP practices and directly apply the principles of abstraction, encapsulation, inheritance and polymorphism.

Additional materials supplied with this assignment include (1) a video illustrating the GUI that should result from this assignment and (2) a set of images that can be used as *icons* to represent the different chess pieces.

1. **(10 points)** Now that you know about the `ArrayList` class, it should be obvious that certain functionality of your program should actually rely on `ArrayList` instead of arrays for better efficiency and convenience. Wherever appropriate, make the corresponding changes in the code.

   The rest of this assignment is about adding a *graphical user interface (GUI)* to your program using the Swing libraries that are part of Java.

2. **(1 points)** Introduce a new package `am.aua.chess.ui` that will encapsulate all the GUI-related aspects of your code.

3. **(20 points)** When the chess board is visualized in a GUI, it is preferable to make the individual board squares *clickable*. This way, one can model a move with two clicks: first one on the origin and the second one on the destination.

   Define a class `BoardSquare` that extends `JButton`—the Swing class for buttons. Introduce two constants for the light and dark colors of squares on the board. Every `BoardSquare` needs to include its x, y coordinates and its color on the board.

   The class should include:

   - a constructor that receives `boolean, int, int` parameters—a flag for the color, x coordinate, and y coordinate;
   - an accessor for the coordinates that returns the pair as an array of length two;
   - a mutator `setPiece(String)` that, given the letter representation of a chess piece (e.g. `"K"`/`"L"` are for white kings, while `"p"` is for black pawn), sets the button icon to the image of the corresponding piece;
   - an overloaded `setPiece()` that sets the button icon to `null`;
   - a mutator `setHighlight(boolean)` that changes the button background to red if the square is highlighted. Otherwise, uses the original (light or dark) color for that square.

4. **(24 points)** A Swing application starts with a desktop window. This functionality is provided by the `JFrame` class, and the easiest way to use it is to extend it. Define a new class `ChessUI` that extends `JFrame`.

   Every `ChessUI` object needs to keep track of the current `Chess` game, a matrix of `BoardSquare` objects, and a `Position` to memorize the click on origin of a move.

   The only, no-arg, constructor should first initialize the instance variables, using a new chess game, set the size of the window, set the title of the window, set the default close operation, choose an appropriate layout for the window. Then, it should set up a `JPanel` to contain the GUI board, add it to the window, and set the window as visible.

   For the `JPanel`, in turn, you should choose an appropriate layout of correct dimensions, set the size of the panel, set up all the `BoardSquare` objects in the matrix and add each one of them onto the panel.

   Note how for the `BoardSquare` objects, it is possible to choose their light/dark color flags based on their indices in the board. Each such button should have a fixed preferred size of square shape. In addition, we want the program to be responsive to clicks on these buttons. Thus, a new *anonymous* action listener needs to be created for each. Its `actionPerformed` method should identify the `BoardSquare` that is the source of the click event, identify the coordinates of that square on the board and use a call to the utility method `boardClicked` (described below) to respond to the click.

   The `private void boardClicked(int[] coordinates)` method needs to consider two cases: first click of a move already made, or no clicks for a move made yet.

   If no clicks for a move have been made yet, an origin position should be created from the given coordinates. All the squares reachable from this origin should become highlighted. Make sure you check in advance that the selected square actually contains a piece of the player whose turn it is now.

   On the other hand, if a click for the move origin has already been made, a destination position should be created from the given coordinates. Now, a move can be attempted between the memorized origin and the newly created destination. After that, we can "forget" the memorized origin and get rid of the highlights on the squares.

   One last `updatePieces` method is required in this class to keep the icons on the `BoardSquare` buttons up-to-date with the current arrangement of chess pieces. Whenever a square doesn't contain a piece, it shouldn't have an icon, and vice versa, an icon should be introduced when a piece is moved onto a square.

   In order to achieve this effect, `updatePieces` should loop through all the positions on the board, and call one of the overloaded `setPiece` methods for each square.

   Insert `updatePieces` invocations in the correct places in the constructor and the `boardClicked` method.

5. **(5 points)** Finally, the `main` method needs to be modified to incorporate the GUI into the program.

   If the program is executed without any *command line arguments*, then a `ChessUI` object should be created. If the program is executed with a single `-console` command line argument, then the console user interface should be used for the program, as before.