Task Description

Create a game which implements the Rubik clock. In this game there are 9 clocks. Each clock can show a time between 1 and 12 (hour only). Clocks are placed in a 3x3 grid, and initially they set randomly. Each of the four clocks on a corner have a button placed between them, so we have four buttons in total. Pressing a button increases the hour on the four adjacent clocks by one. The player wins, if all the clocks show 12. Implement the game, and let the player restart it. The game should recognize if it is ended, and it has to show in a message box how much steps did it take to solve the game. After this, a new game should be started automatically

Plan:

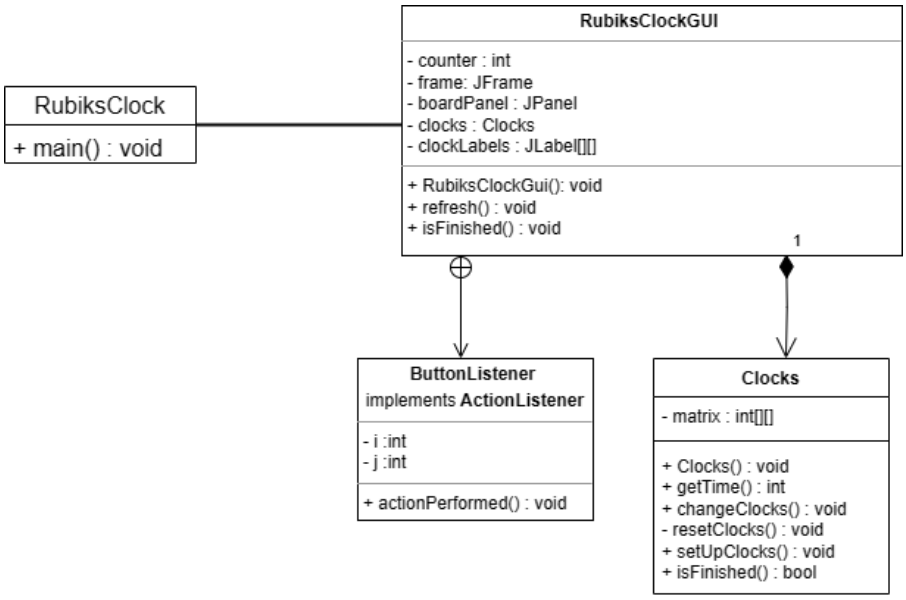
The GUI will be a 5*5 matrix, in which there will be buttons, labels which represent clocks, and empty labels. Let's suppose i is the row index which starts from 0, and j is the column index which also starts from 0. We want each 4 buttons on a corner to have a button placed between them. Hence, we will put the buttons on the coordinates (1,1),(1,3),(3,1), and (3,3). All the other positions for which (i+j) is an even number will be labels which represent clocks. On the other hand, all fields in the grid for which (i+j) is an odd number will be empty labels.

We need to be able to modify the labels which represent clocks, so we will store them in a 3*3 JLabel matrix called labels.

Onto the game logic. Let's create a class which will manage the clocks, and call it clocks. Clocks is represented by a 3*3 integer matrix, with each integer representing the time on a specific clock. Responsibilities of this class include the set-up of clocks, adjusting the time of adjacent clocks when a button is clicked, retrieving individual clock time, and determining game completion.

Whenever a change is made in the Clocks class, we will reflect it onto the labels matrix, and hence to the user.

UML



Description of each method

Getters and setters will not be covered in this section to save space and time.

Clocks.changeClocks()

Changes the time of the clocks based on the clicked button.
It does this by taking the i and j coordinates of the clicked button. Buttons can be located only at (1,1), (3,1), (1,3), and (3,3), so the function does not accept clicks from other coordinates.
Furthermore, this function manages the discrepancy between the GUI matrix (5*5) and its own matrix (3*3).

Clocks.resetClocks()

Resets time of all clocks to 12

Clocks.setUpClocks()

Sets up the clocks to a random time, while ensuring the game remains solvable. It achieves this by setting all the clocks to 12, and then stimulating clicks in different positions. It does this by calling the changeClocks() function with random i and j coordinates.

Buttons can be located only at (1,1) (3,1) (1,3) (3,3), hence the generated random i and j coordinates are either 1 or 3.

Clocks.isFinished()

Checks if the clock puzzle is finished. The puzzle is finished if all clocks show 12.

Clocks.isFinished()

Checks if the clock puzzle is finished. The puzzle is finished if all clocks show 12.

RubiksClockGUI()

Initializes the frame, clocks, graphical components for the game. Sets up the menu bar with options to reset the puzzle and exit the game.

RubiksClockGUI.refresh()

Refreshes the displayed labels on the GUI based on the current state of the clocks.

RubiksClockGUI.isFinished()

Checks if the game is finished and displays a message if it is. If the game is finished, resets the clock class, updates the display, and sets counter to 0.

ButtonListener.actionPerformed()

Changes the clocks when a button is clicked, increments the step counter, refreshes the display based on the clock class, and checks if the game is finished.

Test Cases

1) The game loads with 9 clocks, each four clocks on the corner have a button in the middle.
2) Each clock is assigned a random value as its time. Although random, these values should lead to a solvable solution.
3) Clicking on a button should increase the time by one unit for each of the adjacent clocks.
4) In case the time for a clock goes beyond 12, it should reset back to 0 instead of increasing indefinitely. In other words, available time's for these clocks are natural numbers between 0 and 12 inclusive.
5) Each click on a button is counted and displayed in a message box at the end of the game.
6) Clicking on the menu tab at the top left corner, opens a menu bar with two menu items: reset and exit.
    a. Clicking on the reset button should reload the game, adhering to the test cases above.
    b. Clicking the exit button should exit the game.
7) If "ok" is clicked in the message box at the end of the game, game should be reloaded adhering to the test cases above.