ITI 1121. Introduction to Computing II - Summer 2015

Assignment 3: Implementing a GUI to solve SUDOKU puzzle (100 points, weight 6.25%)

(Last Modified in July 7, 2015)

NOTE: unlike other assignments this assignment can be done in groups of 2 students or individually.

Due date: Monday July 20 at 11:59PM.
The assignment must be uploaded on Virtual Campus by the due date.
Late assignments are accepted between 1 min late up to a maximum of 24 hours late and they will receive a 30% penalty.

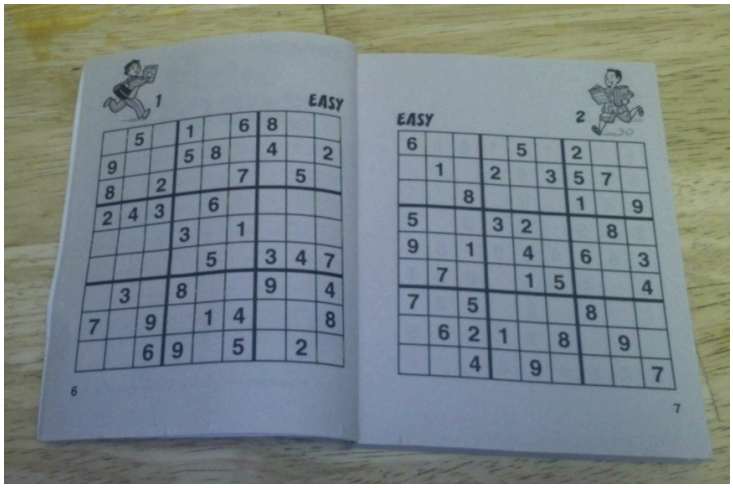
Learning objectives

- Gaining familiarity with concepts involving Graphical User Interfaces and event-driven programming.
- Writing a software application using inheritance, interfaces and object-oriented programming. Graphical user interfaces are a rich example for all of these.
- Designing a simple application using event-driven programming.

For this assignment you will write a simple Graphical User Interface to allow a user to play the Sudoku puzzle.

Sudoku Puzzle

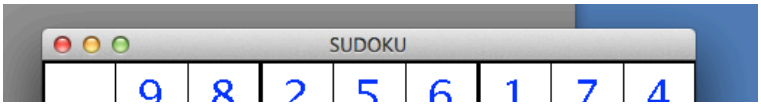
SUDOKU is a placement puzzle in which symbols from 1 to 9 are placed in cells of a 9 x 9 grid made up of nine 3 x 3 subgrids, called regions. The grid is partially filed with some symbols (the "givens"). The grid must be completed so that each row, column and region contains exactly one instance of each symbol. A valid SUDOKU puzzle must have a unique solution; thus for a totally filled SUDOKU grid, not every set of cells chosen to be the "givens" can yield a valid puzzle.

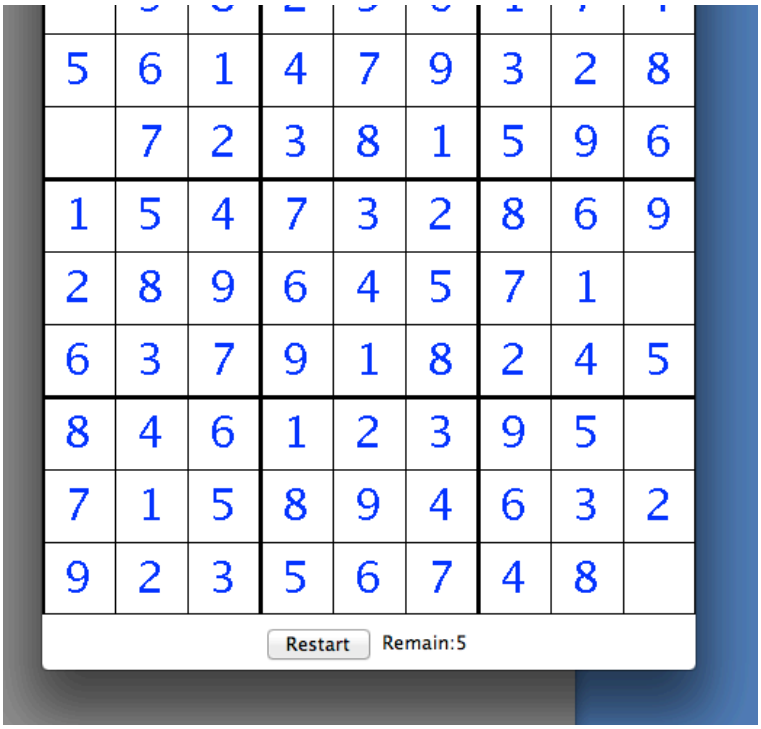


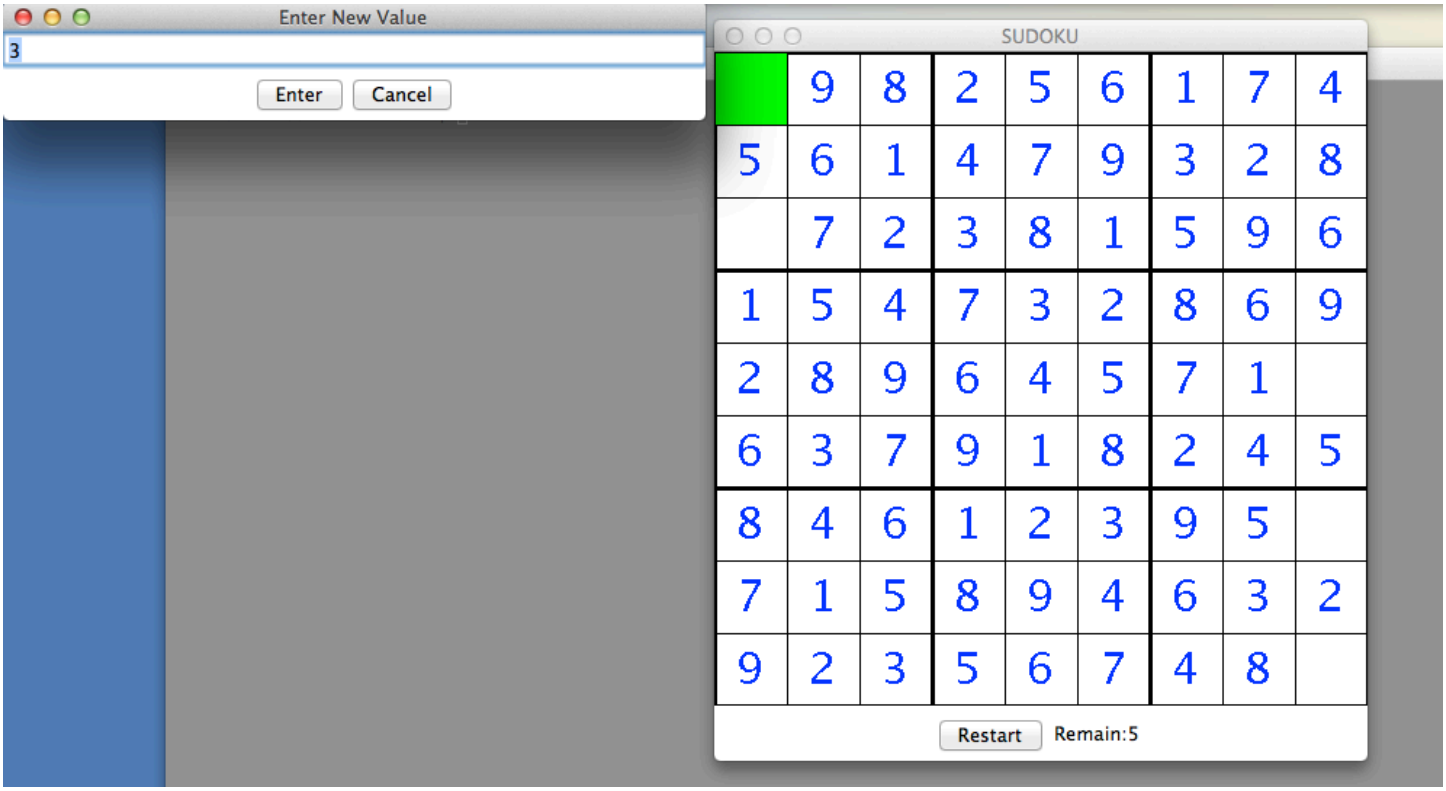
There are many interesting algorithmic questions regarding SUDOKU such as: solving Sudoku puzzles, creating valid puzzles, checking if a set of givens is valid and how many solutions it yields. We are NOT going to concentrate on these questions for this assignment. Instead, we are just going to build a GUI for someone to play SUDOKU. The SUDOKU grid with its givens and the solution of the puzzle are going to be known to the program before hand; the program will prevent the user to complete the grid with wrong numbers by checking the solution.

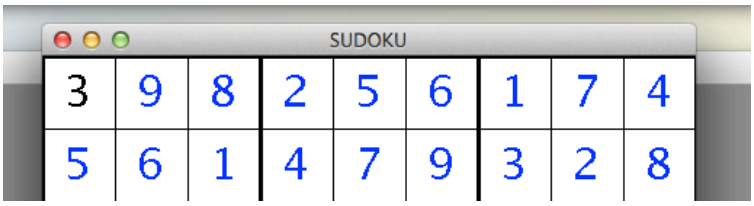
Problem

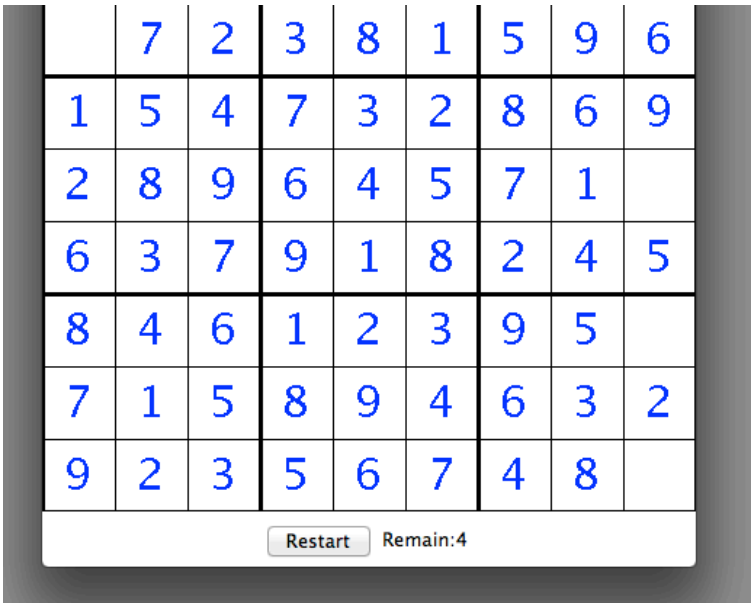
You are implementing a Graphical User Interface (GUI) to allow a user to play Sudoku. Here are some screenshots of our implementation of this GUI:




5 6 1 4 7 9 3 2 8
7 2 3 8 1 5 9 6
1 5 4 7 3 2 8 6 9
2 8 9 6 4 5 7 1
6 3 7 9 1 8 2 4 5
8 4 6 1 2 3 9 5
7 1 5 8 9 4 6 3 2
9 2 3 5 6 7 4 8
Restart
Remain:5


Enter New Value
3
Enter
Cancel
SUDOKU
9 8 2 5 6 1 7 4
5 6 1 4 7 9 3 2 8
7 2 3 8 1 5 9 6
1 5 4 7 3 2 8 6 9
2 8 9 6 4 5 7 1
6 3 7 9 1 8 2 4 5
8 4 6 1 2 3 9 5
7 1 5 8 9 4 6 3 2
9 2 3 5 6 7 4 8
Restart
Remain:5

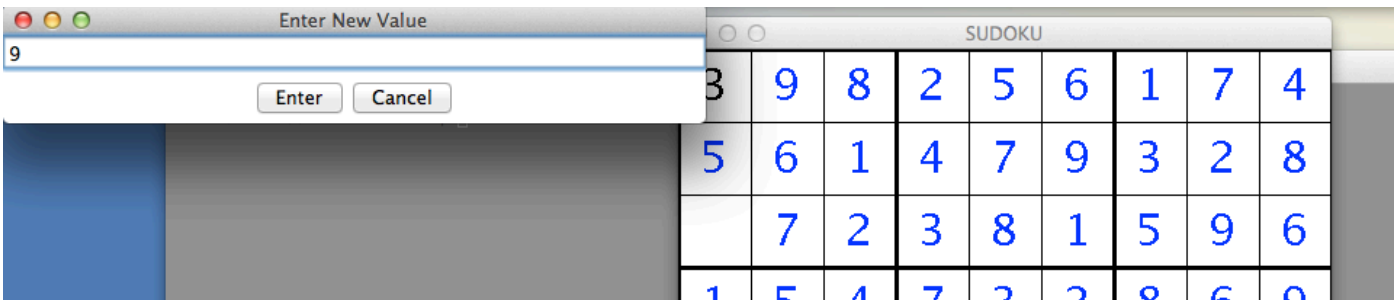

SUDOKU
3 9 8 2 5 6 1 7 4
5 6 1 4 7 9 3 2 8

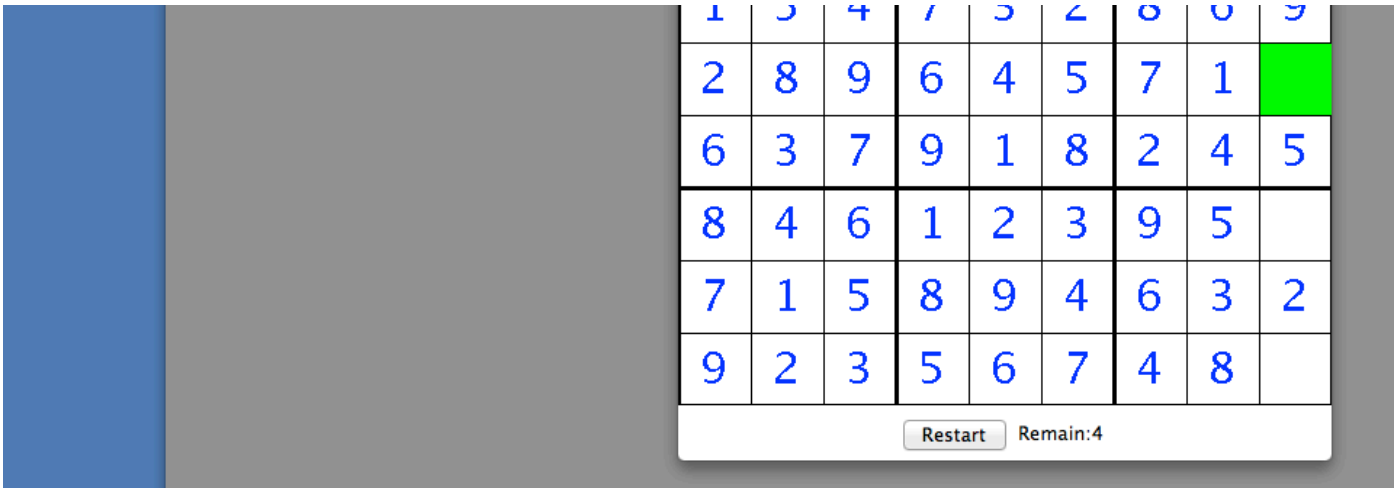


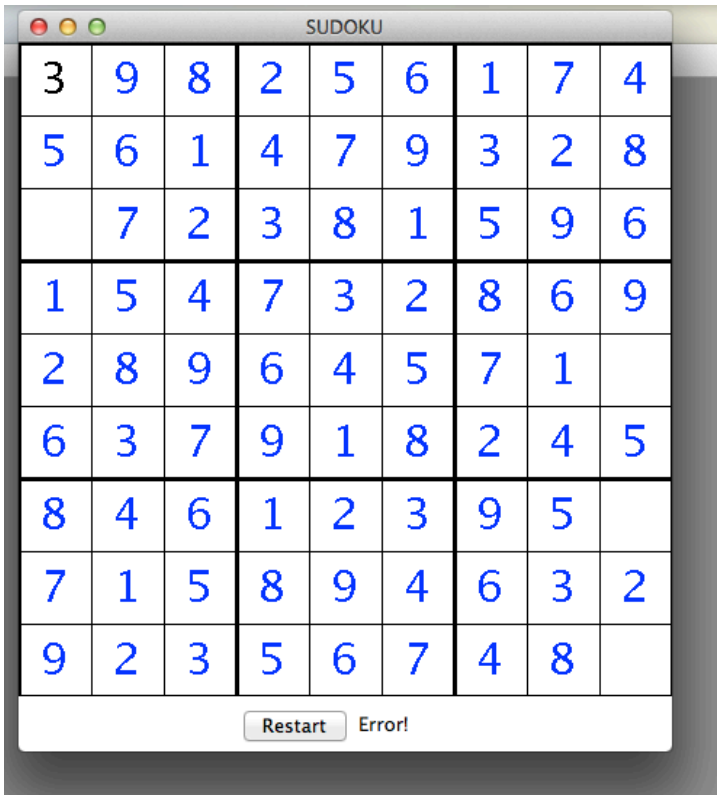
You are given interface SudokuMatrix.java as well as classes SomeSudokuMatrix.java and Test.java.
You must implement classes **SudokuGame** and **SudokuBoard** that will deal with important aspects of the GUI.
Indeed, you may refer back to Laboratory 5 for a comparison or to use as a basis for your code.
SudokuGame is the main GUI holding the frame (in Lab5 refer to GUI.java) that contains a Panel (with a restart Button and a status Label) and an object of a class called SudokuBoard which extends Canvas and is also an ActionListener (in Lab 5 refer to DisplayArea.java).
SudokuBoard is used as a Canvas to draw the board and will also declare a private class that implements a MouseListener. This mouse listener must implement a method "public void mouseClicked(MouseEvent e)" which will retrieve the coordinates of the point where the click was done and identify which Sudoku cell it lies on. After a click a pop-up dialog box must be launched where the user can type the value to be filled in that cell. Note that if the cell is a "given" (number in blue above) or a previously filled value (number in black above), the click should have no effect. In SudokuBoard, the method "public void paint(Graphics g)" of Canvas must be overridden in order to paint the canvas with the current info on the Sudoku board. We suggest the use of java.awt.Graphics2D which extends java.awt.Graphics since it provides a better control over geometry than java.awt.Graphics, for rendering 2-dimensional shapes, texts and images. Class **SudokuBoard** will also have the main logic that controls the filling of cells with numbers, and prevent a cell to be filled with a incorrect number, by holding information about the status of the game, values filled in cells, and having access to the SudokuMatrix object for the puzzle used.

Details and requirements

- The given classes (SudokuMatrix.java, SomeSudokuMatrix.java, Test.java) must be able to communicate with your program without change. In particular, your SudokuGame class must have a constructor with signature: **public SudokuGame(SudokuMatrix puzzle)**
- You must implement the functionality we require, following detailed instructions on functionality below.
- You are free to modify the look of the GUI.
- You are free to use other packages. Our example above that follow Laboratory 5 uses package java.awt. This may be the fastest route for someone with little experience on GUIs since they can rely on the examples from Lab 5, but you may use javax.swing package.
- This assignment can be done individually or in groups of 2 students each.
- **Functionality requirements**:
  - Use different colours for the numbers that are "givens" and the numbers filled by the user.
  - A mouse click on a point inside a Sudoku cell should produce a pop-up dialog box asking for a new value with buttons "Enter" and "Cancel". If either "Enter" or "Cancel" are pressed the dialog box disappears, but we need to process the number entered in the case of "Enter" being pressed.
  - If the number entered in the dialog box is correct (i.e. it matches the number in the Sudoku solution for that cell) then the number must be filled in the cell. The status Label must be updated by decreasing the number of remaining cells to fill (see status "Remain: 4" in the example above.
  - If the number entered in the dialog box is incorrect then it must not be filled in the cell; instead the status message is updated to "Error!". The user can then continue trying to fill cells. This example is illustrated in the pictures below.
  - After all cells are filled, the status message becomes "Solved!"
  - Closing the main frame window should stop the program (like in Lab 5 example).
  - Restart button pressed in the main frame re-initiates the puzzle with the same game.







Files

- SudokuMatrix.java,
- SomeSudokuMatrix.java,
- Test.java
- SudokuGame.java

- SudokuGame.java
- SudokuBoard.java
- Any extra auxiliary classes needed for your interface, for example "startup/MyWindowAdapter.java">MyWindowAdapter.java from Lab5, other auxiliary classes you may or may not decide to create.

Extra Useful Information

We may want to check the following information on specific classes and methods:

- Adding a mouse listener to a component: method addMouseListener
- Interface MouseListener. When creating a class that implements MouseListener, you need to add meaningful code to mouseClicked method only (the other methods need to do nothing).
  For MouseEvent, we found useful methods getX and getY to obtain the coordinate of the point where the mouse was clicked.
- Useful Graphics2D methods to use in method paint() for the Canvas. Your Canvas **paint** method can start as follows:
  public void paint(Graphics g){
  Graphics2D g2 = (Graphics2D)g;
  After this we can use methods of Graphics2D via g2.
  Some methods we found useful: drawRect, drawString, setStroke, setColor, setFont.

Rules and regulations (5/100)

Please follow the general directives for assignments.
You must abide by the following submission instructions:
For assignments of **groups of two** students only one student submits the code; the name of both students must be clearly indicated, also including in StudentInfo display. The other student must submit (by the due date) a text file named "PARTNER.txt" containing the name and student number of the partner who would have already submitted the code.
Include all your java files into a director called **u1234567** where 1234567 is your student id.
Zip this directory and submit via blackboard.
The directory must contain the following files (with your implementation):

- README.txt
- StudentInfo.java
- SudokuMatrix.java (given not to be changed)
- SomeSudokuMatrix.java (given not to be changed)
- Test.java (given not to be changed)
- SudokuGame.java
- SudokuBoard.java
- Any extra auxiliary classes needed for your interface, for example MyWindowAdapter.java from Lab5, and other auxiliary classes you may or may not decide to create.

The assignment is individual or in group of 2 (plagiarism or collaborations towards the solution of the assignment between groups will not be tolerated). Read the information on academic fraud from other assignments.