CS2133: Computer Science II
Assignment 3

Prof. Christopher Crick

1 Message in a Bottle (20 points)

Create a MessageFrame class extending JFrame and a MessagePanel class extending JPanel. Add the panel to the frame. The frame should be titled "Message in a Bottle" and should end the program when the close box is clicked. In the panel, draw a bottle shape using some combination of Graphics object methods like drawLine, drawPolygon, drawPolyline, drawRoundRect and drawArc (look up the Graphics object in the Java API for details). Make the bottle as attractive as you can, although you won't be graded on artistic merit. In the center of the bottle shape, add a message string. Create a Message class with a main method that instantiates the MessageFrame and gets the ball rolling.

2 Sierpinski's Triangle (30 points)

Sierpinski's Triangle is a simple and famous example of a fractal image. It is built recursively from a simple set of rules, illustrated in Figure 2. Your task will be to create an application that illustrates a perfect Sierpinski triangle, regardless of how large the application frame is. As the user moves and resizes the application window, the triangle should be redrawn and made larger or smaller as appropriate.

Your program should display a frame that is based on the size of the user's screen. The paintComponent method of the panel on which you are drawing will be called whenever the frame is resized, so that happens automagically and you don't have to worry about it. JPanels include a getHeight() and getWidth() method that you will be able to use to get the information you need for passing to a recursive draw function that you will write.

The draw algorithm takes the coordinates and dimensions of a square area of the screen as input. If that square is the size of a single pixel, it should call drawRect() on the Graphics object passed into paintComponent, drawing a one-pixel square at the given coordinates. If larger, it should call the draw method three times recursively, once on the lower left quadrant of the square, once on the lower right, and once on an area centered above the other two (as illustrated in Figure 2).

The solution will look a lot like Figure 3, with the largest triangle fitting into the largest square area of the frame, and the smallest triangles being three pixels in size.

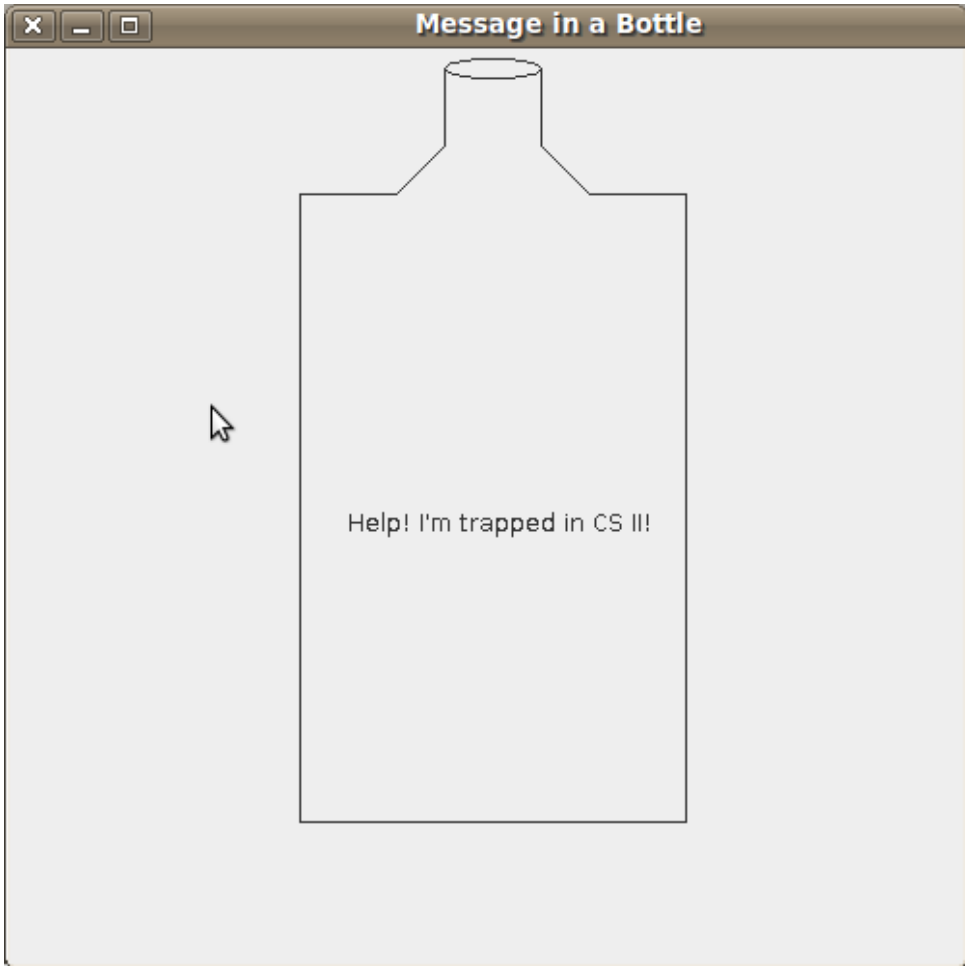

Figure 1: Example output for Problem 1. Feel free to be more creative than this.

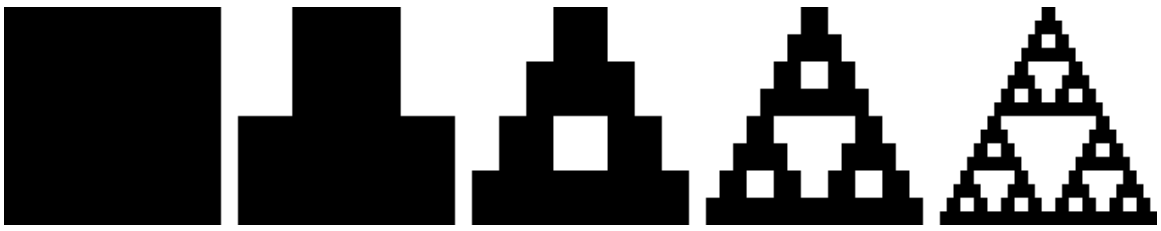

Figure 2: Fractal representation of Sierpinski construction from Problem 2. (Image source: Wikipedia)

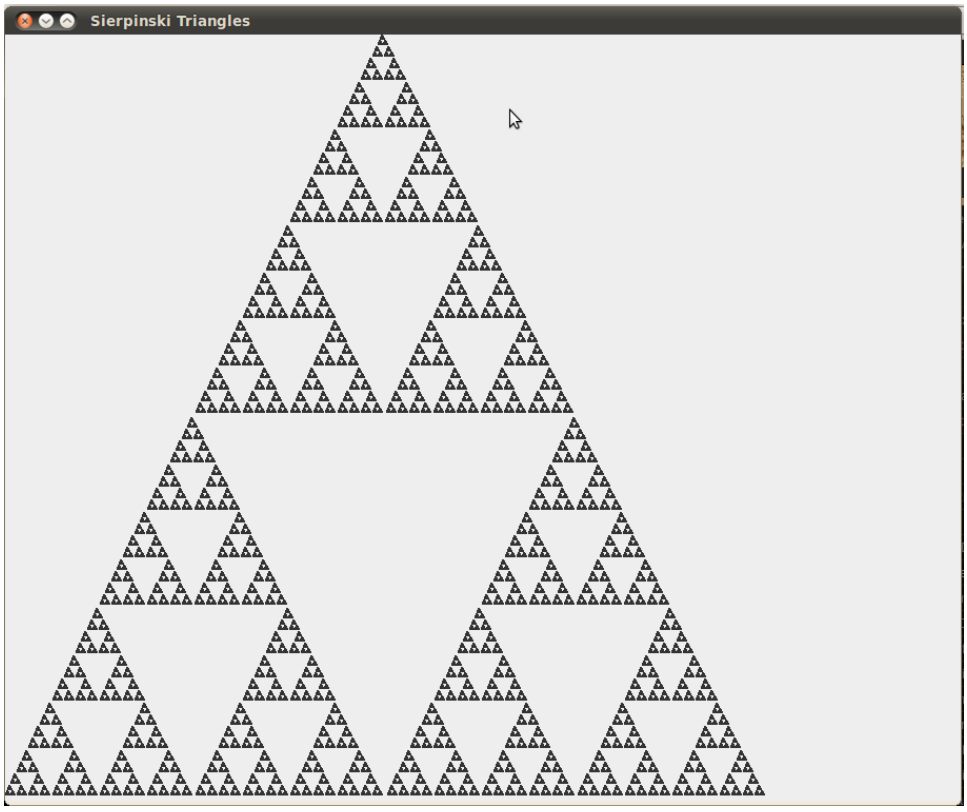

Figure 3: Representative illustration of the Sierpinski problem solution.

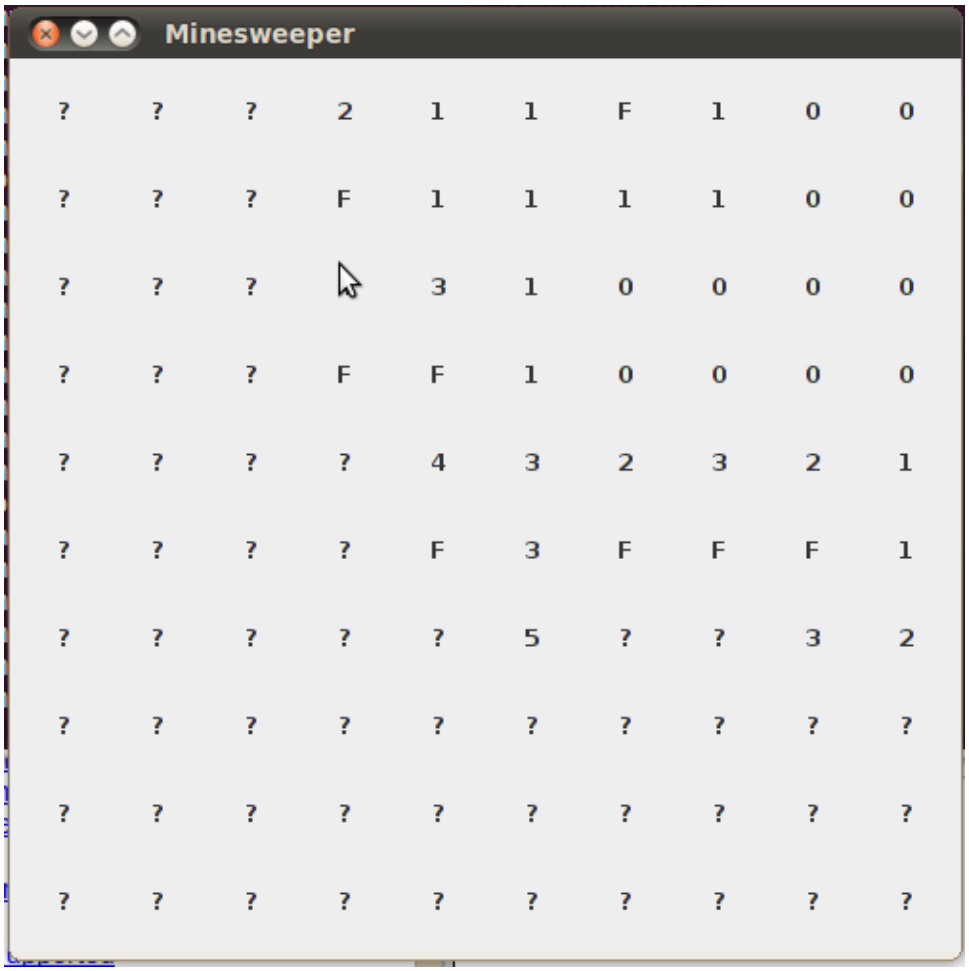

Figure 4: A half-completed game of Minesweeper with a rudimentary design.

3 Minesweeper (50 points)

You are going to write the game of Minesweeper. The Minesweeper board is a graphical grid of squares. A certain number of squares, chosen randomly, conceal dangerous mines. Play proceeds when a user left-clicks on a square. If that square hides a mine, the game is over and the player loses. If not, then stepping on the square reveals the number of mines hidden by squares adjacent to that square – a number between 0 and 8. A player can right-click on a square that has not yet been stepped on to mark it as being potentially mined, and remove the mark with another right-click.

Figure 4 shows a game in progress. Numbers represent squares that have been clicked, 'F's are places the player has placed a flag, and '?'s are squares that have not yet been clicked. This is a singularly unattractive gameboard; you are welcome to add icons or grid squares or any other kinds of graphics to liven up your own version.

You are responsible for designing and implementing the whole game. This is a big task; start early. Here are some suggestions.

- Model-View-Controller architecture! Implement a minesweeper class that plays the game without relying on any GUI elements, instead using method calls for making moves and sending information. This is the model. The paintComponent methods of your UI objects should ask this model what to display. This is the view. The event handlers should call the gameplay methods in the model. This is the controller. Don't let things get mixed up or complicated!

- This is a perfect setting for using a GridLayout. Make it easy to change the number of squares on the board, maybe by setting a static final int.

- JButtons might be a good choice for board squares, but keep in mind that ActionListeners do not know how to respond to right-clicks. You will have to use a MouseAdapter instead. A grid full of JLabels or JButtons that have been extended to have access to the model, and to know where they sit in the grid, would work, as would other more attractive user interface elements.

- When you are setting up your board, you need to randomly determine whether each square is mined or safe. You can select a specific number of squares to be mines, or you can assign each square's status based on a certain probability of being a mine. The latter is easier.

- The class with the main method (Minesweeper.java) and the one extending your JFrame should be only be a couple of lines. All of the heavy GUI lifting will probably be split between the class that extends JPanel and the one that extends whatever you decide to use to represent individual grid squares. Your MouseListener can belong to either of these, but it's probably easier if each individual square is given its own MouseListener. And the most important object, the model that actually implements the game, shouldn't involve GUI at all!

- Make sure the program ends in a satisfying manner when the user clicks on a bomb.

Extra Credit: Because it is always safe to click every mine surrounding a square with a '0' in it, most commercial versions of the game will automatically do so, and then do it again for any new '0' squares uncovered. This can reveal large swaths of safe territory with a single click. Implement this functionality.

Extra Credit: In the header area of the JFrame's BorderLayout, implement a counter showing how many mines remain to be found. This should initially report the number of mines on the board, and decrement or increment whenever the player plants or removes a flag.

Extra Credit: Wow us with the graphic design of your game.

Turning in

This is the first assignment where the design of the programs, and thus the Java files you will require, have been left up to you. The files with the main() function that actually execute the programs, however, should be named Message.java, Sierpinski.java and Minesweeper.java, so we know which ones we're supposed to run. Those three, plus all of the other .java files that define the classes you need for these programs – your subclassed JFrames and JPanels and the like – should be wrapped up in a zip file called assignment_3_your_name.zip and uploaded to the Dropbox at

oc.okstate.edu. Ensure that everything can be compiled and run from the command line. This assignment is due Wednesday, September 28, at noon.