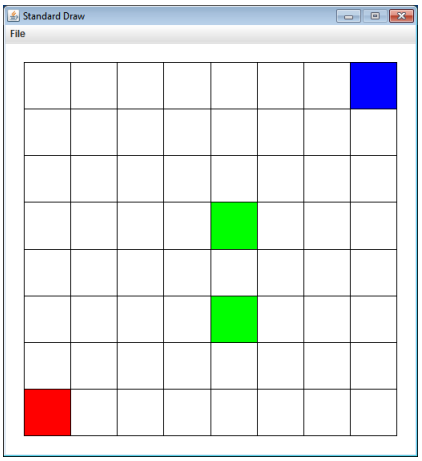
CSCI 136 Programming Exam #1
Fundamentals of Computer Science II
Spring 2012

This part of the exam is like a mini-programming assignment. You will create a program, compile it, and debug it as necessary. This part of the exam is open book and open web. You may use code from the course web site or from your past assignments. When you are done, submit all your Java source files to the Moodle exam #1 dropbox. Please ***double check you have submitted all the required files***.

You will have 100 minutes. No communication with any non-staff members is allowed. This includes all forms of real-world and electronic communication.

Grading. Your program will be graded on correctness and to a lesser degree on clarity (including comments) and efficiency. You will lose a substantial number of points if your code does not compile or crashes on typical inputs.

Overview. You will be implementing a class that simulates paint dripping down a two-dimensional colored grid. The 2D grid has an initial configuration of colored grid boxes. These fixed initial boxes will deflect paint that drips down on them. Here is what an initial grid looks like that has a width of 8 and a height of 8 with initial a red box at (0, 0), a blue box at (7, 7), and green boxes at (4, 2) and (4, 4):



You will be completing the instance methods in the class `Grid`. The class already has some instance variables declared. The `draw()` method is also completely implemented, you should not need to modify `draw()`. Note that our draw() method assumes uncolored grid locations contain the value `null`.

As usual (0, 0) will be the lower-left corner. Start by downloading the file located at:

http://katie.mtech.edu/classes/csci136/grid.zip

The exam is split into three parts, by the end you will have developed the following API:

```
public class Grid
-------------------------------------------------------------------------------
        Grid(int width, int height)      // part1
  Color getColor(int x, int y)           // part1
   void setColor(int x, int y, Color c)  // part1
    int getWidth()                       // part1
    int getHeight()                      // part1
    int getMaxColumn()                   // part1
        Grid(String filename)            // part2
   void drip(int x, int y, Color c)      // part3
   void draw()                           // implemented for you
```

Part 1. In this part, you will be implementing the methods needed to create an empty grid, set the color at a given location, get the grid width, get the grid height, get the color at a given location, and calculate which column has the most colored grid locations.

The exact behavior of each method is described in the comments before the method declaration in the provided stub code. We have given you various chunks of test code for each part of the exam in the `main()` method of `Grid.java`. We will not be grading your `main()` method.

The part 1 test code should produce the above grid graphic and the following console output:

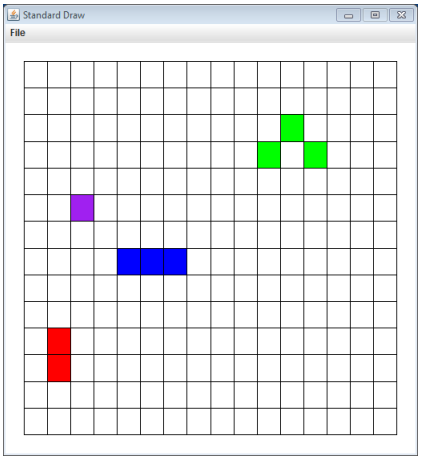
```
Grid size = 8 x 8
Color @ (0,0) = java.awt.Color[r=255,g=0,b=0]
Color @ (1,1) = null
Max column = 4
```

Part 2. Add the ability to load the initial grid configuration from a text file. Create an overloaded version of the `Grid` constructor that takes a single parameter that specifies the filename. Here is the example `grid1.txt` file:

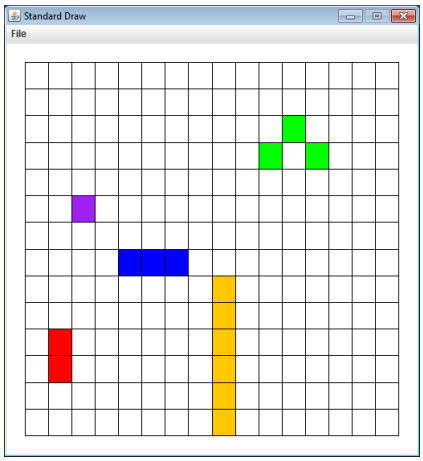
```
16 14
1 3 1.0 0.0 0.0
1 2 1.0 0.0 0.0
10 10 0 1 0
11 11 0 1 0
12 10 0 1 0
4 6 0 0 1
5 6 0 0 1
6 6 0 0 1
2 8 0.6274 0.1254 0.9411
```

The first two numbers give the width (16) and the height (14) of the grid. You can assume any file given to your constructor has two positive integers representing the width and height. After the width and height, 0 or more colored grid locations are specified. Each colored grid location is specified by two integers followed by three floating-point values. For example, the second line in the above specifies a block at x-location 1, y-location 3, with a red color (the RGB value is 1.0, 0.0, 0.0). You can assume all grid locations and RGB colors in the file are valid.

Creating a `Grid` object with the file `grid1.txt` and then calling `draw()` should result in the following:

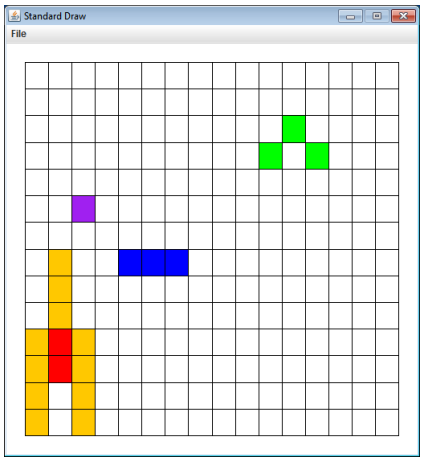


Part 3. A drop of paint is started at a specified (x, y) position in the grid. The drop proceeds down the grid coloring empty grid locations as it goes. So for example, initializing the grid with `grid1.txt`, if a drip is started at (8, 5) with a color of orange, it would color orange the grid locations (8,5) (8,4) (8,3) (8,2) (8,1) and (8,0):



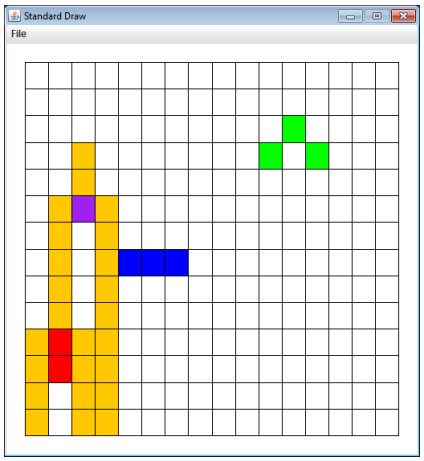
3a

However, if while proceeding down the screen, the grid location directly below a drip's current location is already colored, the drop splits into two parts. The first part attempts to continue down and to the left of the current drip's location. The second part attempts to continue down and to the right of the current drip's location. So for example, loading `grid1.txt` and creating an orange drip at (1, 6) results in:



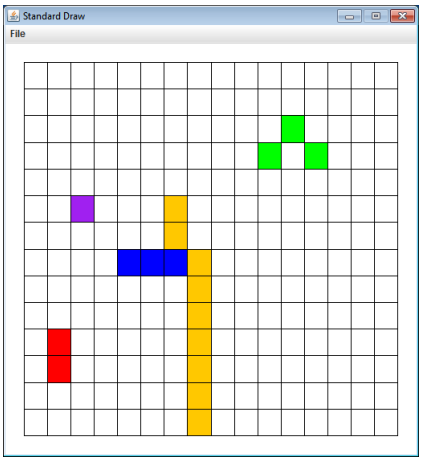
3b

During its journey down the grid, a drop may get split several times. For example, a drip started at (2, 10) splits once when it hits the purple block at (2,8) and again when it hits the red block at (1,3):



3c

It is possible when the drip splits, one or both sides will be unable to proceed due to an existing colored grid location(s). For example, in the grid shown below, a drip started at (6,8) split when it hit the blue horizontal surface at (6, 6). But only the right part of the split was able to continue at (7, 6). The left part was unable to continue since (5, 6) was already colored blue:



3d

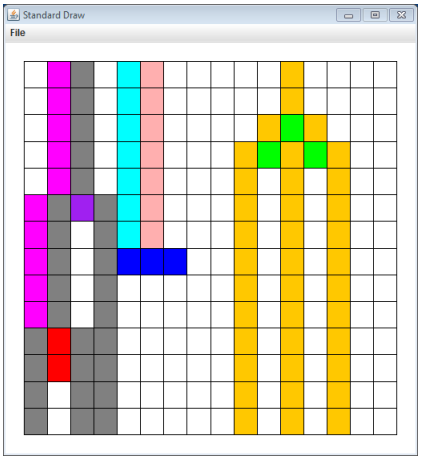
Implement a recursive method to color the grid according to the previous outlined rules. The method's signature is:

```
public void drip(int x, int y, Color c)
```

The recursive algorithm has two base cases:

- The (x, y) location is out of range with respect to the grid.
- The (x, y) location is already colored. This includes being colored by the initial configuration file or by having been colored by a previous drip.

The recursive step should proceed straight down or split if the grid immediately below the current location is already colored. Here is the output from the final test code provided in our `main()` method consisting of 6 calls to `drip()`. Note only 5 drips appear since the last one was created on top of a previous drip:



Console output:

```
Max column = 11
```