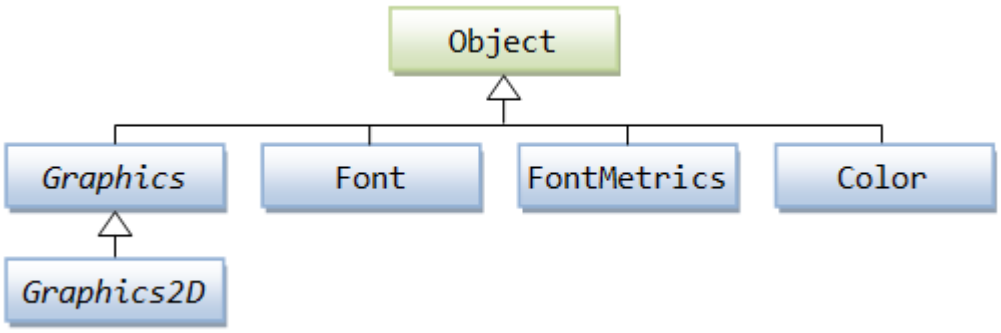
CSCI165 Computer Science II
Lab Assignment
Java 2D Graphic and Matrix processing

Java provides a rich library for drawing custom graphics to the screen.

The java.awt.Graphics Class: Graphics Context and Custom Painting

A *graphics context* provides the capabilities of drawing on the screen. The graphics context maintains states such as the color and font used in drawing, as well as interacting with the underlying operating system to perform the drawing. In Java, custom painting is done via the ***java.awt.Graphics*** class, which manages a graphics context, and provides a set of *device-independent* methods for drawing texts, figures and images on the screen on different platforms.

The java.awt.Graphics is an abstract class, as the actual act of drawing is system-dependent and device-dependent. Each operating platform will provide a subclass of Graphics to perform the actual drawing under the platform, but conform to the specification defined in Graphics. I know that we haven't gotten into classes and inheritance yet so don't be too concerned about the details here. You are going to build a template that can be used for drawing.



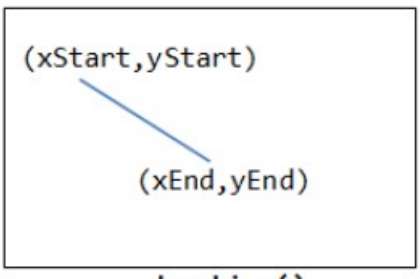
Graphics Class Drawing Methods

The Graphics class provides methods for drawing three types of graphical objects:

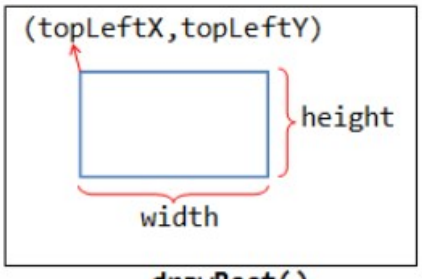
1. **Text strings:** via the drawString() method. Take note that System.out.println() prints to the *system console*, not to the graphics screen.
2. **Vector-graphic primitives and shapes:** via methods drawXxx() and fillXxx(), where Xxx could be Line, Rect, Oval, Arc, PolyLine, RoundRect, or 3DRect.
3. **Bitmap images:** via the drawImage() method.

```
// Drawing (or printing) texts on the graphics screen:
drawString(String str, int xBaselineLeft, int yBaselineLeft);

// Drawing lines:
drawLine(int x1, int y1, int x2, int y2);
drawPolyline(int[] xPoints, int[] yPoints, int numPoint);

// Drawing primitive shapes:
drawRect(int xTopLeft, int yTopLeft, int width, int height);
drawOval(int xTopLeft, int yTopLeft, int width, int height);
drawArc(int xTopLeft, int yTopLeft, int width, int height, int startAngle, int arcAngle);
draw3DRect(int xTopLeft, int, yTopLeft, int width, int height, boolean raised);
drawRoundRect(int xTopLeft, int yTopLeft, int width, int height, int arcWidth, int arcHeight)
drawPolygon(int[] xPoints, int[] yPoints, int numPoint);

// Filling primitive shapes:
fillRect(int xTopLeft, int yTopLeft, int width, int height);
fillOval(int xTopLeft, int yTopLeft, int width, int height);
fillArc(int xTopLeft, int yTopLeft, int width, int height, int startAngle, int arcAngle);
fill3DRect(int xTopLeft, int, yTopLeft, int width, int height, boolean raised);
fillRoundRect(int xTopLeft, int yTopLeft, int width, int height, int arcWidth, int arcHeight)
fillPolygon(int[] xPoints, int[] yPoints, int numPoint);

// Drawing (or Displaying) images:
drawImage(Image img, int xTopLeft, int yTopLeft, ImageObserver obs)
drawImage(Image img, int xTopLeft, int yTopLeft, int width, int height, ImageObserver o);
```

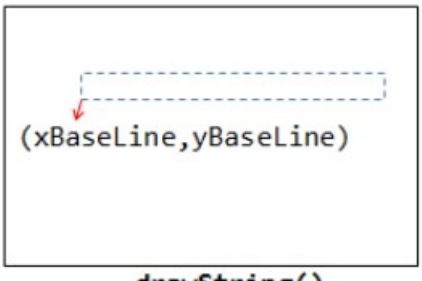
These drawing methods are illustrated below. The drawXxx() methods draw the outlines; while fillXxx() methods fill the internal. Shapes with negative ***width*** and ***height*** will not be painted.



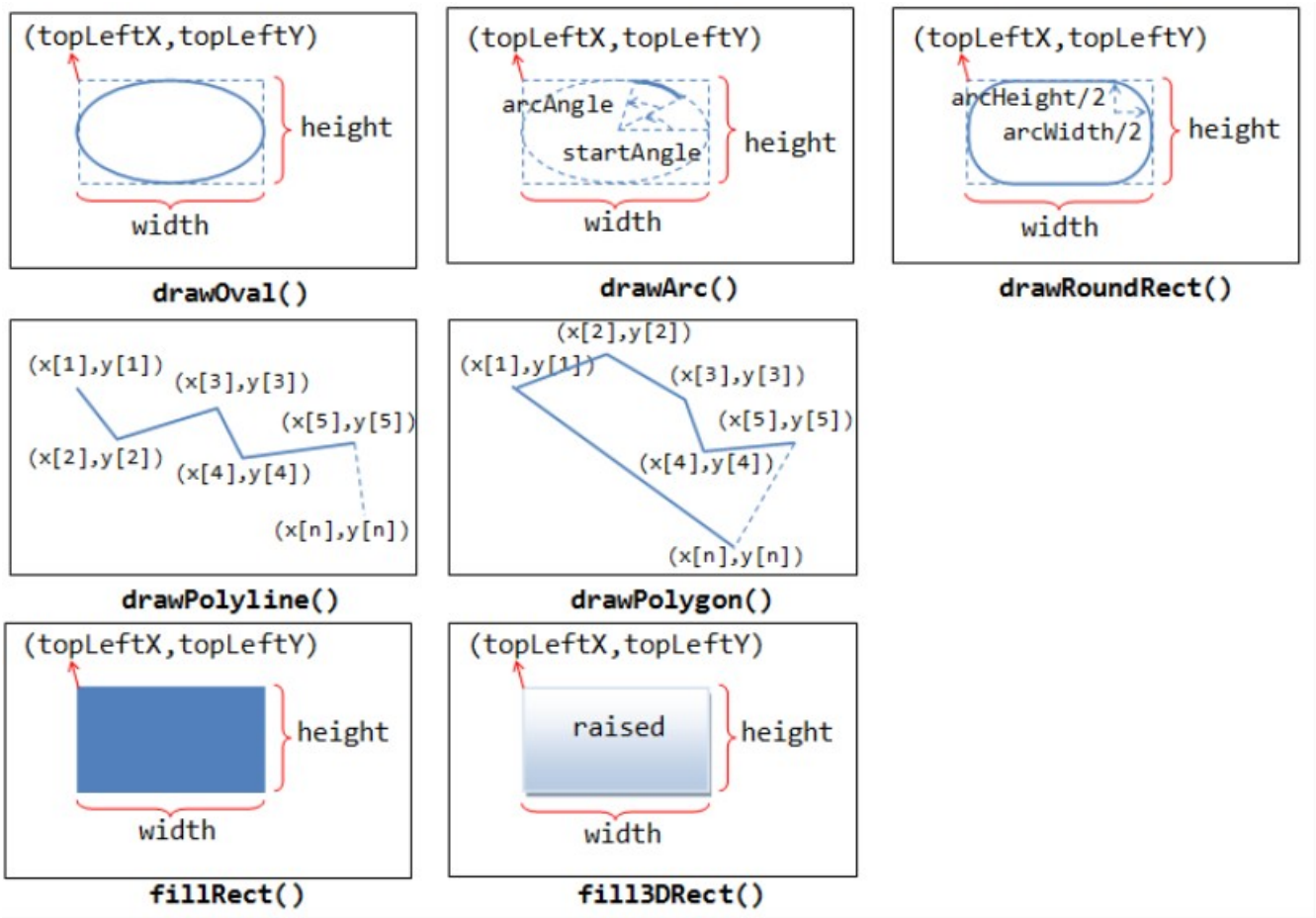
drawLine()



drawRect()



drawString()



Graphics Class' Methods for Maintaining the Graphics Context

The graphic context maintains *states* (or *attributes*) such as the current painting color, the current font for drawing text strings, and the current painting rectangular area (called *clip*). You can use the methods getColor(), setColor(), getFont(), setFont()

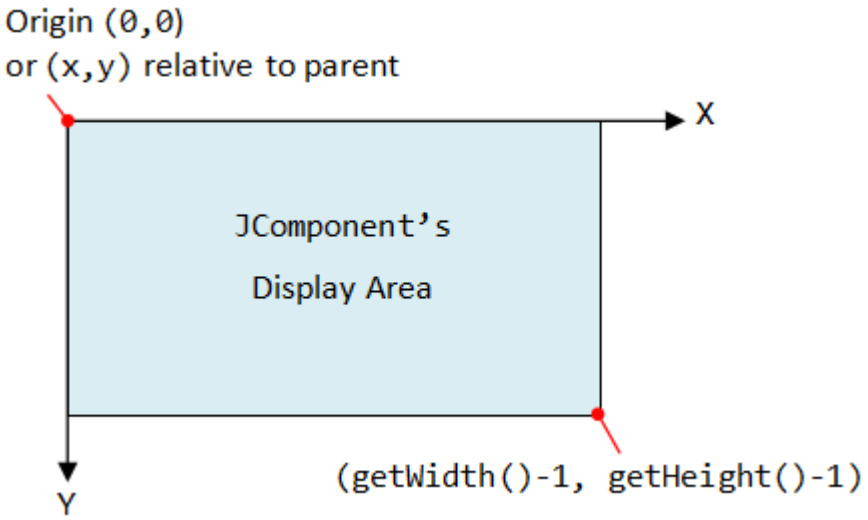
```
// Graphics context's current color.
void setColor(Color c)
Color getColor()

// Graphics context's current font.
void setFont(Font f)
Font getFont()

// Clear the rectangular area
void clearRect(int x, int y, int width, int height)

// Copy the rectangular area to offset (dx, dy).
void copyArea(int x, int y, int width, int height, int dx, int dy)

// Translate the origin of the graphics context to (x, y).
// Subsequent drawing uses the new origin.
void translate(int x, int y)
```

Graphics Coordinate System

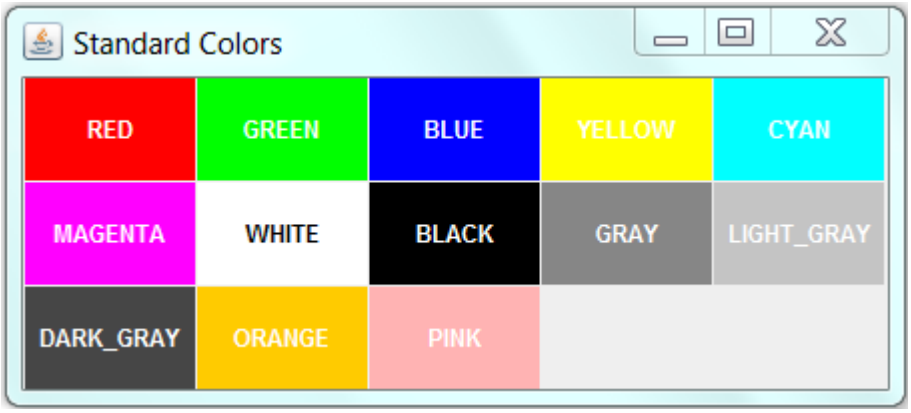
In Java Windowing Subsystem (like most of the 2D Graphics systems), the origin (0,0) is located at the top-left corner.

Each component/container has its own coordinate system, ranging for (0,0) to (width-1, height-1) as illustrated.

You can use method getWidth() and getHeight() to retrieve the width and height of a component/ container. You can use getX() or getY() to get the top-left corner (x,y) of this component's origin relative to its parent.



java.awt.Color



The class java.awt.Color provides 13 standard colors as named-constants. They are: Color.RED, GREEN, BLUE, MAGENTA, CYAN, YELLOW, BLACK, WHITE, GRAY, DARK_GRAY, LIGHT_GRAY, ORANGE, and PINK. (In JDK 1.1, these constant names are in lowercase, e.g., red. This violates the Java naming convention for constants. In JDK 1.2, the uppercase names are added. The lowercase names were not removed for backward compatibility.)

You can use the toString() to print the RGB values of these color (e.g. System.out.println(Color.RED)):

```
RED        : java.awt.Color[r=255, g=0,   b=0]
GREEN      : java.awt.Color[r=0,   g=255, b=0]
BLUE       : java.awt.Color[r=0,   g=0,   b=255]
YELLOW     : java.awt.Color[r=255, g=255, b=0]
MAGENTA    : java.awt.Color[r=255, g=0,   b=255]
CYAN       : java.awt.Color[r=0,   g=255, b=255]
WHITE      : java.awt.Color[r=255, g=255, b=255]
BLACK      : java.awt.Color[r=0,   g=0,   b=0]
GRAY       : java.awt.Color[r=128, g=128, b=128]
LIGHT_GRAY: java.awt.Color[r=192, g=192, b=192]
DARK_GRAY : java.awt.Color[r=64,  g=64,  b=64]
PINK       : java.awt.Color[r=255, g=175, b=175]
ORANGE     : java.awt.Color[r=255, g=200, b=0]
```

You can also use the RGB values or RGBA value (A for alpha to specify transparency/opaque) to construct your own color via constructors:

```
Color(int r, int g, int b);             // between 0 and 255
Color(float r, float g, float b);       // between 0.0f and 1.0f

// alpha of 0 for totally transparent, 255 (or 1.0f) for totally opaque
// The default alpha is 255 (or 1.0f) for totally opaque
Color(int r, int g, int b, int alpha);         // between 0 and 255
Color(float r, float g, float b, float alpha); // between 0.0f and 1.0f


// For example:

Color myColor1 = new Color(123, 111, 222);
Color myColor2 = new Color(0.5f, 0.3f, 0.1f);
Color myColor3 = new Color(0.5f, 0.3f, 0.1f, 0.5f);  // semi-transparent
```

Graphics Drawing Template

The file ***Drawer.java*** (in the lab directory) contains everything you need to draw some primitive Java 2D graphics. The method **doDrawing** is where you will place all of your drawing code. For the programming assignment for this module you will need to define additional methods to perform matrix processing. Those methods can be defined in here and called from the **doDrawing** method.

```
33      // add your own drawing intructions in this method
34      private void doDrawing(Graphics g) {
35
36          /*
37              RGB Colors:
38              ================================================
39              Black       => (0, 0, 0)        => low elevation
40              Mid Grey    => (128, 128, 128)  => mid elevation
41              White       => (255, 255, 255)  => high elevation
42
43              Grey Scale colors are scaled in matching set of 3 numeric values
44          */
45
46          Graphics2D g2d = (Graphics2D) g;
47
48          //<=====> ADD YOUR DRAWING CODE HERE <==========>//
49
50
51      }
```

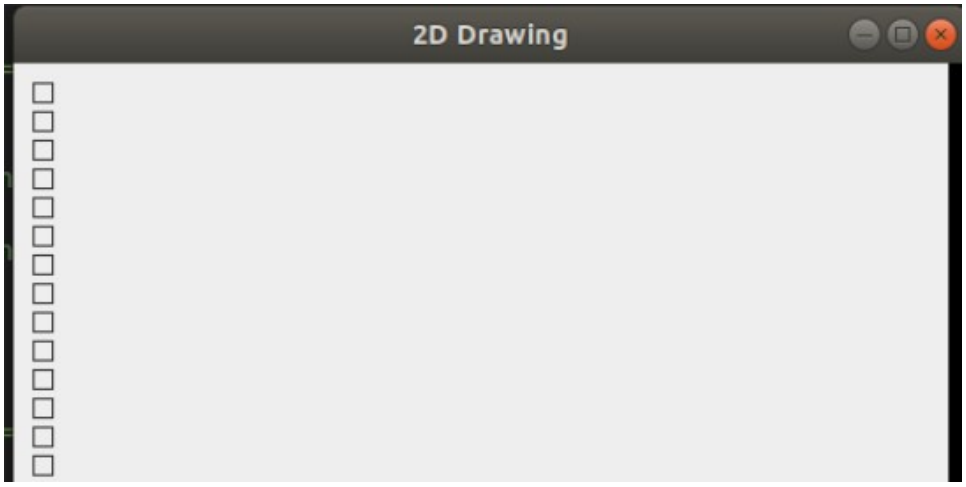
Lab Tasks: Modify the Drawer.java file and add the following code. You need to follow along. Feel free to be as creative as you'd like though. These steps are just the bare minimum.

1) Draw a column of hollow rectangles with default color

Compile and run

```
    // add your own drawing intructions in this method
    private void doDrawing(Graphics g) {

        /*
            RGB Colors:
            ================================================
            Black       => (0, 0, 0)        => low elevation
            Mid Grey    => (128, 128, 128)  => mid elevation
            White       => (255, 255, 255)  => high elevation

            Grey Scale colors are scaled in matching set of 3 numeric values
        */

        Graphics2D g2d = (Graphics2D) g;

        //<=====> ADD YOUR DRAWING CODE HERE <==========>//
        int x = 10, y = 10;
        for(int i = 0; i < 30; ++i){
            g2d.drawRect(x, y, 10, 10);
            y += 15;
        }
    }
```

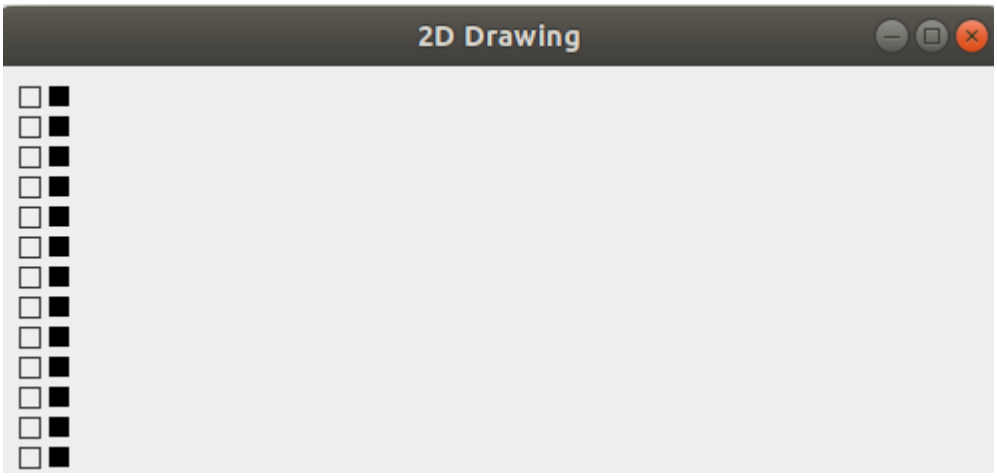
You should see a window appear (JFrame) with a column of 30 hollow rectangles of size 10 x 10. I'm only showing a portion here, to conserve space



2) Now let's change the RGB color of the graphics context and print another column of 30 10 x 10 rectangles. This time filled in. Move the X coordinate over so the shapes don't overlap

```
55          // RGB (0, 0, 0) is black
56          g2d.setColor(new Color(0, 0, 0));
57          y = 10; x += 15;
58          for(int i = 0; i < 30; ++i){
59              // fillRect draws a filled in rectangle
60              g2d.fillRect(x, y, 10, 10);
61              y += 15;
62          }
```

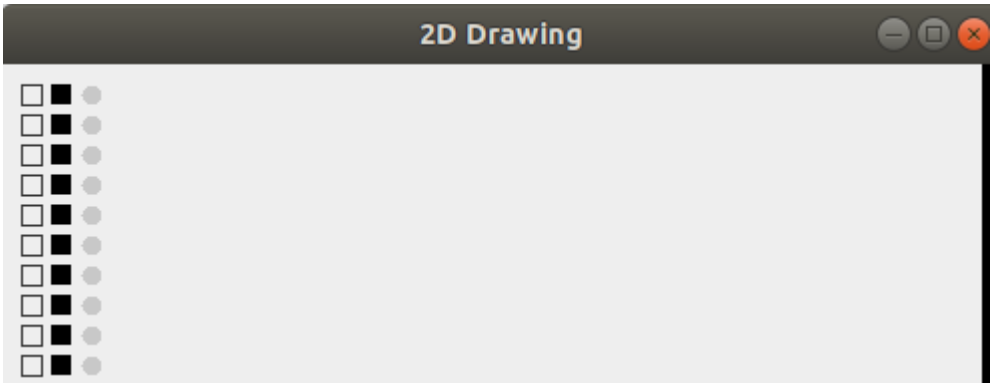
Compile and run. Again, I'm only showing a portion.



Let's keep going

3) Draw a column of filled in Ovals, colored grey. Don’t forget to move the X coordinate

```
64        g2d.setColor(new Color(200, 200, 200));
65        y = 10; x += 15;
66        for(int i = 0; i < 30; ++i){
67            g2d.fillOval(x, y, 10, 10);
68            y += 15;
69        }
```



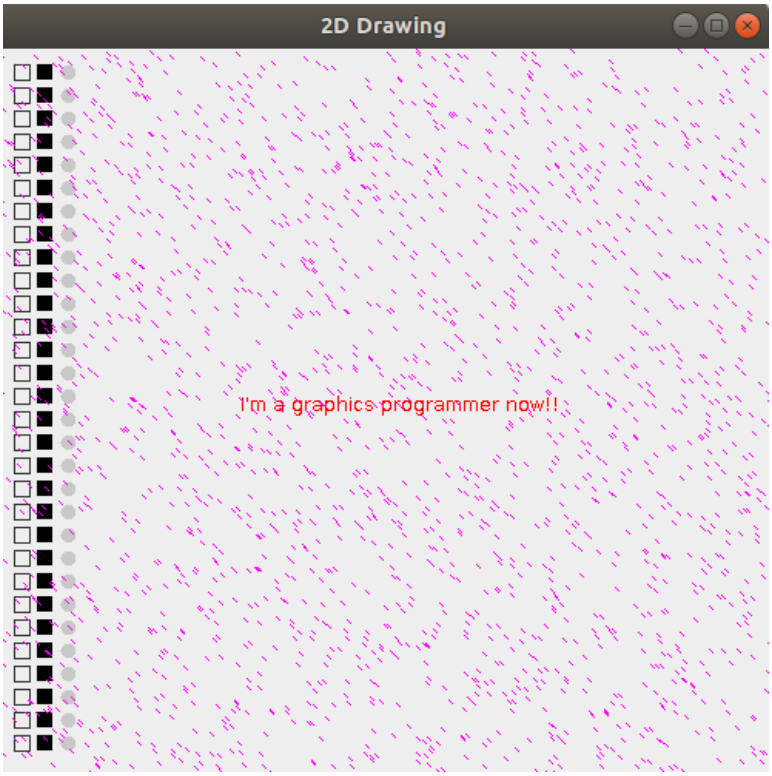
Ok . . . let’s try something different. Let’s draw a red string of text somewhere near the middle of the window. Use one of the provided constants for the color, instead of RGB. Use the drawString method

```
71        int width = getWidth();      // get width of the window
72        int height = getHeight();    // get height of the window
73        g2d.setColor(Color.RED);     // use a constant to change the color
74        String s = "I'm a graphics programmer now!!";
75        // roughly calculate the middle of the window
76        g2d.drawString(s, (width / 2) - s.length() * 3 , height / 2);
```

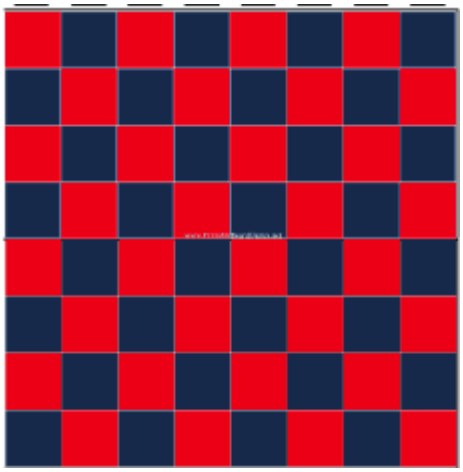
Compile and run

And for the final demonstration. Let's draw 2000 random magenta colored tick marks.

```
78          g2d.setColor(Color.MAGENTA);
79          Random random = new Random();
80          for(int i = 0; i < 2000; i++){
81              x = Math.abs(random.nextInt()) % width;
82              y = Math.abs(random.nextInt()) % height;
83              g2d.drawLine(x, y, x + 2, y + 2);
84          }
```



TASK: Create a copy of the ***Drawer.java*** file, change the name to ***ChessBoard.java*** Add the code to draw a chess board resembling the following. Use whatever colors you'd like. Change the size of the Jframe (setSize method) to a square and use math to calculate the chessboard square size and location.



Part Two Matrix processing

Create a file called ***MatrixStuff.java***

1) In the main method define a 2D array called matrix of size 50 x 20. Write the following methods

1. **public static void fillArrayRowMajor(int[][] matrix)** Fill array in row-major order with data from ***number_list.txt***
2. **public static int findMax(int[][] matrix)** Maximum value in matrix
3. **public static int findMin(int[][] matrix)** Minimum value in matrix
4. **public static int findMaxOfRow(int[][] matrix, int row)** Find the maximum value of **row** in **matrix**
5. **public static int findMinOfRow(int[][] matrix, int row)** Find the minimum value of **row** in **matrix**
6. **public static int findMaxOfColumn(int[][] matrix, int column)** Find the maximum value of **column** in **matrix**
7. **public static int findMinOfColumn(int[][] matrix, int column)** Find the minimum value of **column** in **matrix**
8. **public static void fillArrayColumnMajor(int[][] matrix)** Fill array in column-major order with data from ***number_list.txt***
9. **public static void printRow(int[][] matrix, int row, int num_cols)** print matrix[row] in num_cols columns
10. **public static int smallestChange(int[][] matrix)** return index of the row that experiences the smallest amount of change from element to element. Looking at adjacent cells
    1. positive change: values increase . . . array[i] > array[i + 1]
    2. negative change: values decrease . . . array[i] < array[i + 1]
11. In the main method that demonstrate that each of these methods functions correctly. Include descriptive messages with each output.
12. Using a copy of the ***Drawer.java*** called ***ShowMinMax.java*** program draw a grid of size 50 x 20 using rectangle of size 10 x 10. The rectangles should be filled with RGB (128, 128, 128). This grid will map to the data in your matrix. Color the minimum value of each row GREEN. Color the maximum value of each row RED

Submission: Push modified **Drawer.java**, **ChessBoard.java, ShowMinMax.java** and **MatrixStuff.java** to your repository. Do not push class files.