Lab 06: Randomized Skyline

In this lab, we'll play around with Java's psuedorandom number generator.
Specifically, we're going to draw a skyline using random numbers. Also, this lab uses `Graphics2D`, which is a more modern graphics library than what we've used in previous labs. Note that, for our purposes, not much changes. However, you are encouraged to look at: https://docs.oracle.com/javase/7/docs/api/java/awt/Graphics2D.html and experiment with what drawing capabilities are available.

To begin:

First, copy and paste the following code into `Skyline.java`:

```
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Dimension;
import javax.swing.JPanel;
import javax.swing.JFrame;
import java.util.Random;
import java.awt.Graphics2D;
import java.awt.geom.Rectangle2D;
import java.awt.RenderingHints;
import java.awt.GradientPaint;

public class Skyline extends JPanel{
    public static final int WIDTH=1024;
    public static final int HEIGHT=768;
    public static void main(String[] args){
        JFrame frame = new JFrame("Skyline");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setContentPane(new Skyline());
        frame.pack();
        frame.setVisible(true);
    }
    public Skyline(){
        this.setPreferredSize(new Dimension(WIDTH, HEIGHT));
    }

    @Override
    public void paintComponent(Graphics gOri){
        Graphics2D g = (Graphics2D) gOri;
        g.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
```

```
        RenderingHints.VALUE_ANTIALIAS_ON);

        GradientPaint sunSet= new GradientPaint(0, 0, Color.BLACK, 0, HEIGHT, new Color(55, 0, 0));
        g.setPaint(sunSet);
        g.fill(new Rectangle2D.Double(0, 0, WIDTH, HEIGHT));

        Random rand = new Random();
        //Your code here

    }
}
```

To compile:

`javac Skyline.java`

To run:

`java Skyline`

To compile and run (if there are no compilation errors):

`javac Skyline.java && java Skyline`

Notes about Random:

To generate a random number, we first need a `Random` object (`Random rand = new Random();`, in the code, above). This is our pseudorandom number generator. From it, we can generate random numbers by calling `rand.nextInt(n)` which returns an `int` in the set `{0, 1, 2, 3, … n-1}`. So, for example: `rand.nextInt(3)` returns either `0`, `1`, or `2`.
See:
https://docs.oracle.com/javase/7/docs/api/java/util/Random.html
for other available methods.

To complete this lab:

You should draw a skyline with a horizon, stars, and a star cluster, as described as follows.

The Horizon

To draw the horizon, pick a random number between 95 and 105. That is the height of the horizon for the left most edge of the screen. Now go across the columns (of pixels) of the image, make the height of that column be the height of the previous column plus a random number between -5 and 5.

Stars

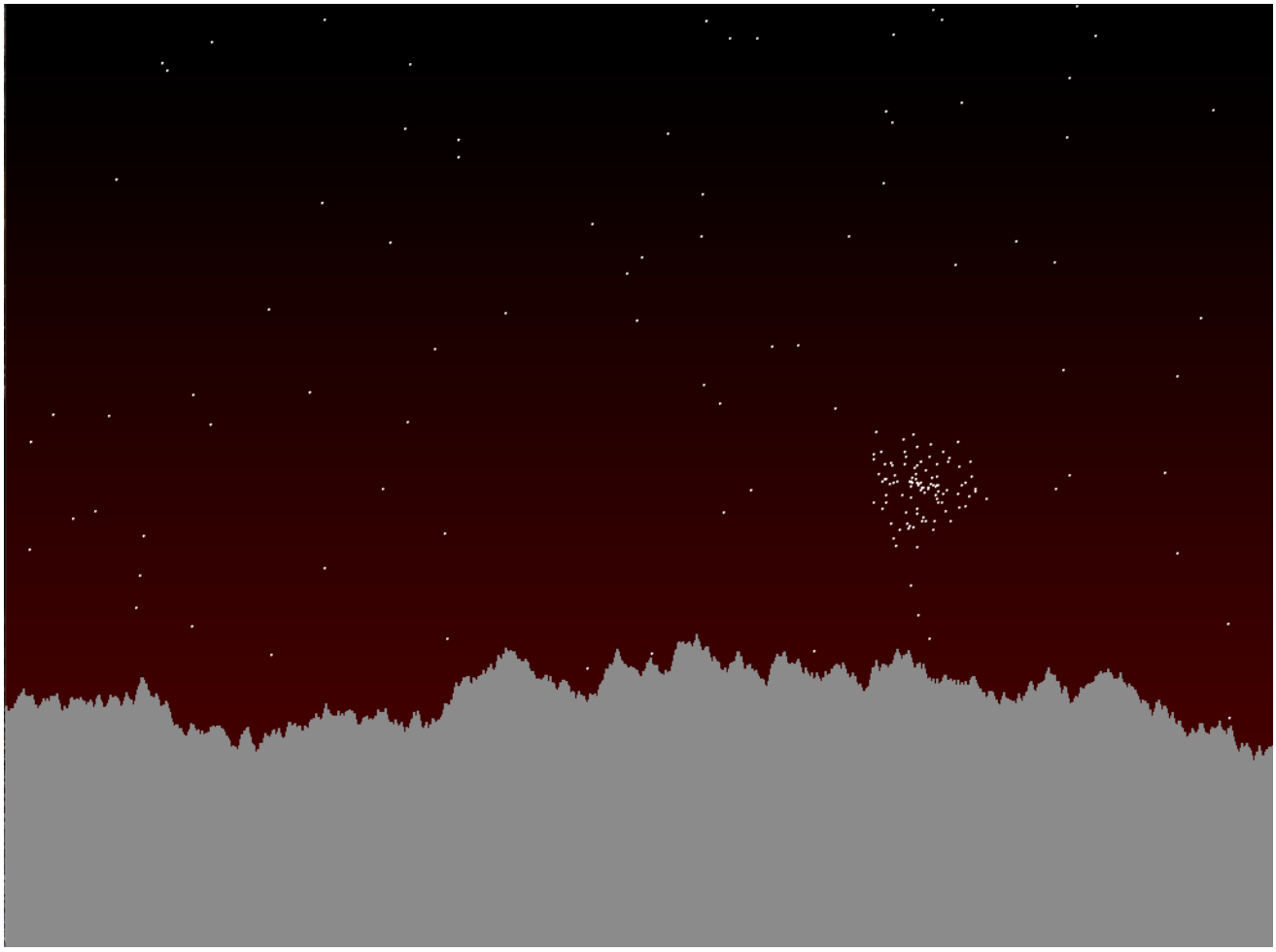
Place roughly 100 stars randomly throughout the sky.

A Star Cluster

Pick a random location roughly in the middle of the sky. This will be the center of our cluster. Draw 100 stars around that center point. The cluster should be roughly circular, rather than a square. To accomplish this, use the `rand.nextGaussian()` method, which draws a floating point from $\mathcal{N}(0, 1)$. See:
https://docs.oracle.com/javase/7/docs/api/java/util/Random.html#nextGaussian()
and
https://en.wikipedia.org/wiki/Normal_distribution
for more information.

Note: `rand.nextGaussian()` will return a very small floating point method. To make your star cluster reasonably large, you should scale it up.

In the end, your skyline should look like some artistic version of the following, minimalistic interpretation:



If you want a challenge:

1. After running your code, resize the window. That's annoying, isn't it? The reason this is happening is because we're generating our random numbers inside the `paintComponent` method, which gets called every time the window is resized. Fix that, so that the image remains static even as `paintComponent` is repeatedly called.
2. Make it pretty. Instead of dots, have the starts be more artistic.
3. Make it animated! Have the stars twinkle and add random shooting stars.
4. Use java.awt.Graphics2D to add all sorts of fancy. For example, have the star cluster be darker in the center and lighter near the edges. Or make the sunset gradient look more sunset like.

What you should submit:

`Skyline.java`

And remember the survey: https://cosc112s22.page.link/Lab06Survey