Informatik 2 Group 2

Lab 7: Fun with Calculators 2

Bartholomäus Berresheim (s0568624) & Lars Vorswerg (571683)

Index

Introduction

Finger exercises

1. What exactly is an equilateral triangle? Can you write a class that...
2. What is the mathematical formula for finding the midpoint of a...
3. What is the resolution of your computer screen? How can you find...
4. Briefly describe what a **Sierpinski Triangle** is.

Lab exercises

1. First set up a Window that can handle drawing. Can you get the...
2. Once you can draw the triangle, now draw a triangle that...
3. Expand your triangle drawing algorithm to draw in a specific color...
4. Fill the middle triangle on each step with an appropriate color...

Reflection

Bartholomäus
Lars

Code

Introduction

In this week's lab, we worked with JWindows/JFrames to create a Sierpinski Triangle. This was also the first lab in which we recursively called methods.

Bartholomäus Berresheim 568624
Lars Vorwerg 571683

Finger exercises

1. What exactly is an equilateral triangle? Can you write a class that draws a triangle? What data do you need to know in order to put a triangle at a particular position on the screen?

And equilateral triangle, is a triangle whose sides are all of equal length and each of its corners has a 60° angle. It is possible to write a class that draws a triangle, and the date needed to put a triangle at a particular position would be to know the resolution of the screen and the x&y coordinates of where you want to put the triangle.

2. What is the mathematical formula for finding the midpoint of a line segment that connects two Points?

The mathematical formula corresponds to xM = (x1+x2)/2 and yM = (y1+y2)/2.

xM/yM are the coordinates of the midpoint, while x1/y1 and x2/y2 correspond to the coordinates of the connected points.

3. What is the resolution of your computer screen? How can you find out? What is the largest equilateral triangle that you can show on a screen with this resolution?

The resolution of the computer screen is 1920x1080. You can find it out by browsing through the pc settings. The largest equilateral triangle that we can make with this resolution is one whose sides have a length of 1080 pixels.

4. Briefly describe what a Sierpinski Triangle is.

A sierpinski Triangle is an equilateral Triangle subdivided recursively into smaller equilateral Triangles.

Lab exercises

1. First set up a Window that can handle drawing. Can you get the Window to draw an equilateral triangle? What is the largest one you can get on the screen?

We started by creating a class called Triangle that extends JPanel, in which we then added a main() method. In it we first created a JWindow called “window”,

then we added a new Object of type Triangle to the window, after that we set the size of the window to the size of our screen and we set its visibility to true.

```
public class Triangle extends JPanel {

    public static int width = 1920;
    public static int height = 1080;


public static void main(String[] args) {

    JWindow window = new JWindow();
    window.getContentPane().add(new Triangle());
    window.setSize(width,height);
    window.setVisible(true);
```

To draw the triangle, we did two things. The first things was that we created a class called Point in which we created 2 private int fields x and y, followed by a constructor that has two ints as input, which he then sets as value for the 2 fields, and after the constructor we added a getter for each of the fields.

```
public class Point {

    private int x;
    private int y;

    Point(int x, int y){
        this.x = x;
        this.y = y;
    }
    public int getX() {
        return x;
    }
    public int getY() {
        return y;
    }
}
```

The second thing we did was to add an int field called "h" and to create a method called paint, that has Graphics as input, to the Triangle class. In the method we created 3 objects of type Point called "p1", "p2" and "p3" respectively, followed by an if-else statement that checks whether the height of the window is bigger or equal to the width, and depending on the case, it sets the values of the points in a way that the base of the triangle is on the smaller one (height or width). Doing it this way, allows us to always create the biggest equilateral triangle possible for the current window sizing.

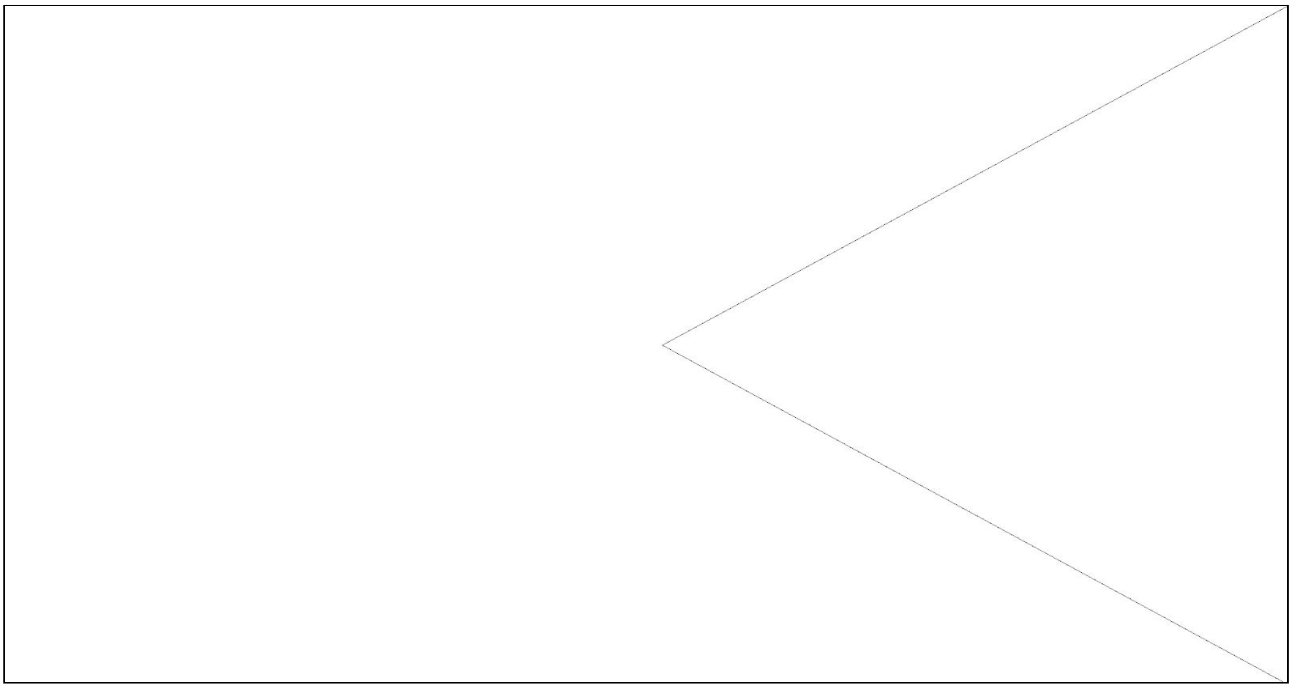
```
private static int h;
```

```
public void paint(Graphics g) {

    Point p1;
    Point p2;
    Point p3;

    if (height >= width) {
        p1 = new Point (0,height);
        p2 = new Point (width,height);
        h = (int) (height-(width/2*Math.sqrt(3)));
        p3 = new Point (width/2,h);
    }else {
        p1 = new Point (width,0);
        p2 = new Point (width,height);
        h = (int) (width-(height/2*Math.sqrt(3)));
        p3 = new Point (h,height/2);
    }
```

Afterwards, we created 2 int arrays xCoords and yCoords that respectively contained the x and y coordinates of the 3 points. Using the arrays, we then called the method drawPolygon() to draw the triangle.

```
int xCoords[] = {p1.getX(), p2.getX(), p3.getX()};
int yCoords[] = {p1.getY(), p2.getY(), p3.getY()};
g.drawPolygon(xCoords, yCoords, 3);
```

Now once executed, we get this :



(the border was added afterwards to get a contrast between the screenshot and the page.)

Each of the Triangle sides has a length of 1080 pixels, which corresponds to the largest possible equilateral triangle possible with this resolution.

2. Once you can draw the triangle, now draw a triangle that connects the midpoints of each of the lines. You now have 4 triangles. For each of the three outer triangles, recursively draw a triangle that connects the midpoints. What is your termination condition, what is the measure?

For this task we first created a method called getMidpoint. It gets 2 Objects of type Point as input using which it calculates the midpoint of those 2 points, as described in the 2nd prelab exercise, and then returns said midpoint.

```
public Point getMidpoint(Point p1, Point p2) {

  Point p3 = new Point((p1.getX()+ p2.getX())/2,(p1.getY()+ p2.getY())/2);
  return p3;

}
```

Once finished, we continued by creating a method called drawTriangle(), that gets 5 inputs, 3 of which being Objects of type Point, another being an Object of type Graphics2D and one int that represents the number of recursions we want. We then moved the 2 previously mentioned arrays and the calling of the drawPolygon() method (from the paint() method), into an if statement contained in the drawTriangle() method, that checks whether the number of recursion is equal to 0. If true, it draws the triangle.

```
public void drawTriangle(Graphics2D g, Point p1, Point p2, Point p3, int recursion) {

  if (recursion == 0) {

  int xCoords[] = {p1.getX(), p2.getX(), p3.getX()};
  int yCoords[] = {p1.getY(), p2.getY(), p3.getY()};
  g.drawPolygon(xCoords, yCoords, 3);
```

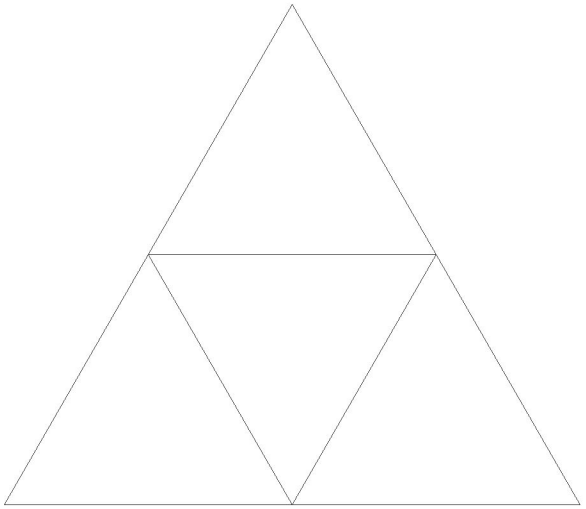
If not, it goes to an else-statement in which we created 3 new Points, each having the midpoints of one of the triangles sides as value. Followed by 3 callings of the drawTriangle() method, using one of the inputted points, its corresponding midpoints and the number of recursion-1 as input.

```
}else {

Point midP12 = getMidpoint (p1, p2);
Point midP13 = getMidpoint (p1, p3);
Point midP23 = getMidpoint (p2, p3);

drawTriangle(g, p1, midP12, midP13, recursion-1);
drawTriangle(g, midP12, p2, midP23, recursion-1);
drawTriangle(g, midP13, midP23, p3, recursion-1);
```

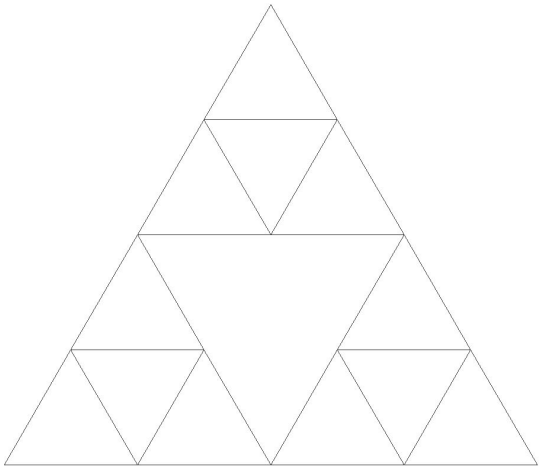
To finish things of, we added an Object of type Graphics2D to the paint() method, and set its value to the Graphics g castet into an Object of type Graphics2D. Followed by a calling of the drawTriangle() method.

```
  Graphics2D g2 = (Graphics2D) g;
  drawTriangle(g2, p1, p2, p3, 1);
}
```

Before executing it, we changed the size of the window for convenience sake and we moved the base of the triangle away from the border, so that we can see it better. Now once we execute it, we get:



If we change the number of recursions to 2, we get:



This means that the termination condition of our program is that the int recursion is equal to/or hits 0. Once that is the case, it won't go into the else statement from the drawTriangle() method.

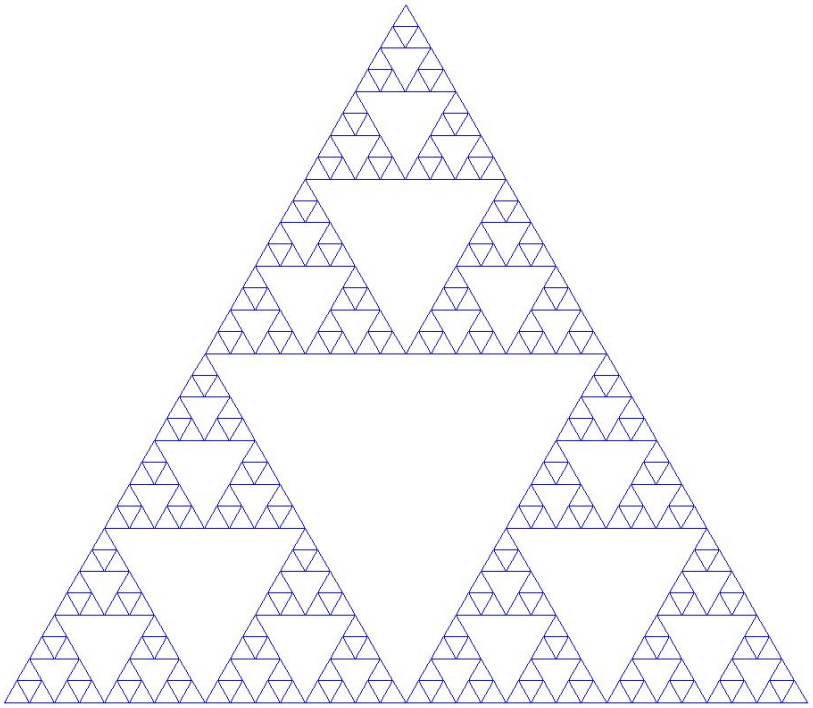
3 & 4 Expand your triangle drawing algorithm to draw in a specific color. Choose a different color for every level of the algorithm.

Fill the middle triangle on each step with an appropriate color. Choose the size of the first triangle depending on what size the window is. Redraw the triangle when the window is resized.

To draw the triangle in a specific color, we added a calling of the setColor() method at the beginning of the drawTriangle() method.

```
g.setColor(Color.blue);
```

Now, once we execute the program, we get:



While working on the part about drawing the different triangles in varying colors, we ran into some issues, which is why we moved on to the coloring part of the 4th exercise, before coming back to fix this issue.

For the filling of the middle triangles, we first created a new Color c in the drawTriangle() method. We created it in a way that its rgb values increase or

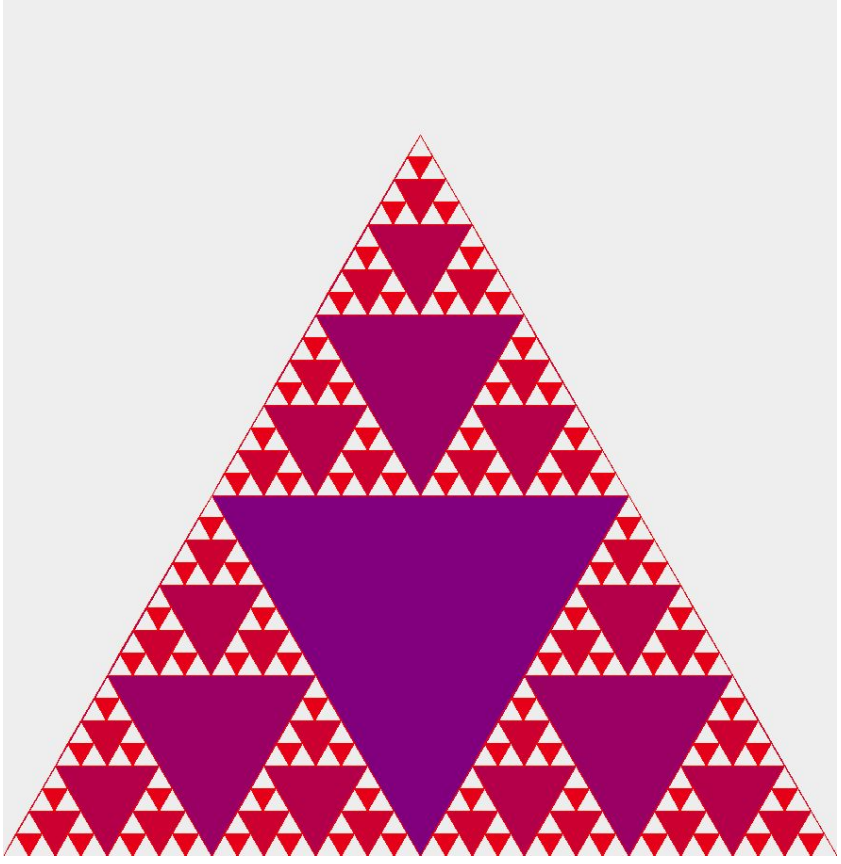
decrease depending on the recursion number. After that, we set it as the color of "g".

```
Color c = new Color(255-recursion*25,0,recursion*25);
  g.setColor(c);
```

Afterwards, we added 2 new int arrays to the else statement. They respectively contain the x and y Coordinates of the "midP12", "midP13" and "midP23" Points. Using these arrays, we then called the fillPolygon() method.

```
int xxCoords[] = {midP12.getX(), midP13.getX(), midP23.getX()}
int yyCoords[] = {midP12.getY(), midP13.getY(), midP23.getY()}
g.fillPolygon(xxCoords, yyCoords, 3);
```

Now once executed, we can see the following:



(It may not be discernible on the screenshot but all the triangle borders have the same color, which as mentioned before is wrong).

Since the triangle was already taking the size of the window into account, we didn't have to change anything there.

The first thing we did for the resizing part, was to change the JWindow into a JFrame or else we couldn't actively change the size of the window, while the

program was running. And we also added a calling of the setDefaultCloseOperator() method, or else the program doesn't get terminated once we close the window.

```
JFrame frame = new JFrame();
frame.getContentPane().add(new Triangle());
frame.setSize(width,height);
frame.setVisible(true);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

Afterwards, we tried working with ComponentEvents to get the triangle repainted but that didn't work out for some unknown reason. After some deliberating, we chose to ask Juri and he told us about the use of the Dimension type Object. Which is why, we first added a new field of type Dimension called "D" with the fields width and heights as parameters.

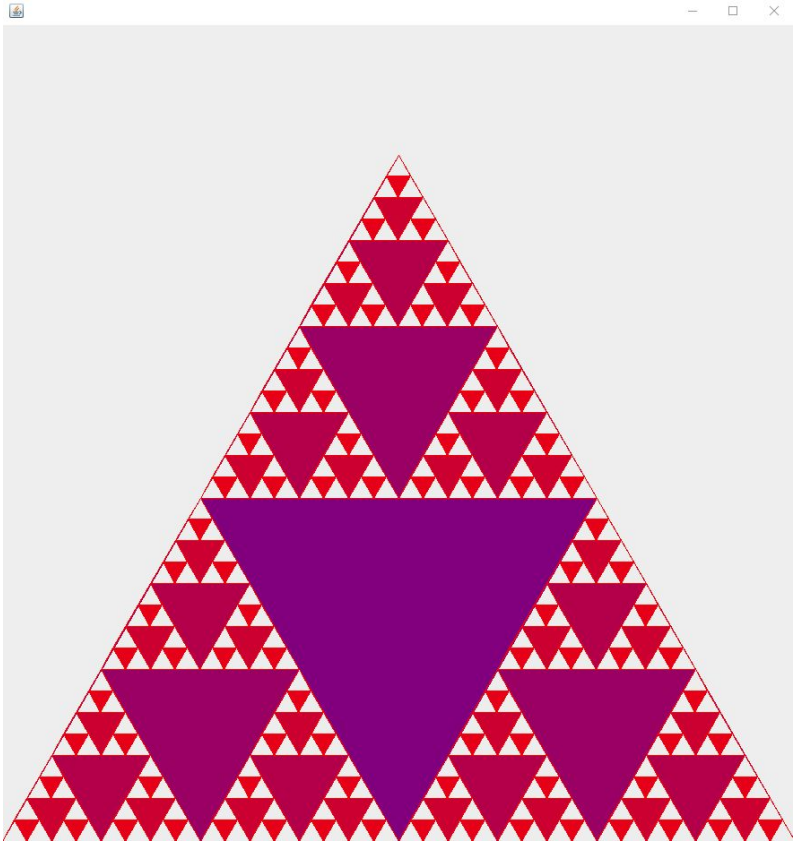
```
private Dimension D = new Dimension(width, height);
```

We then added a calling of the setSize() method using D, that had getParent().getSize() as input, in the paint() method. After that, we changed the values of width and height to D.width and D.height respectively.

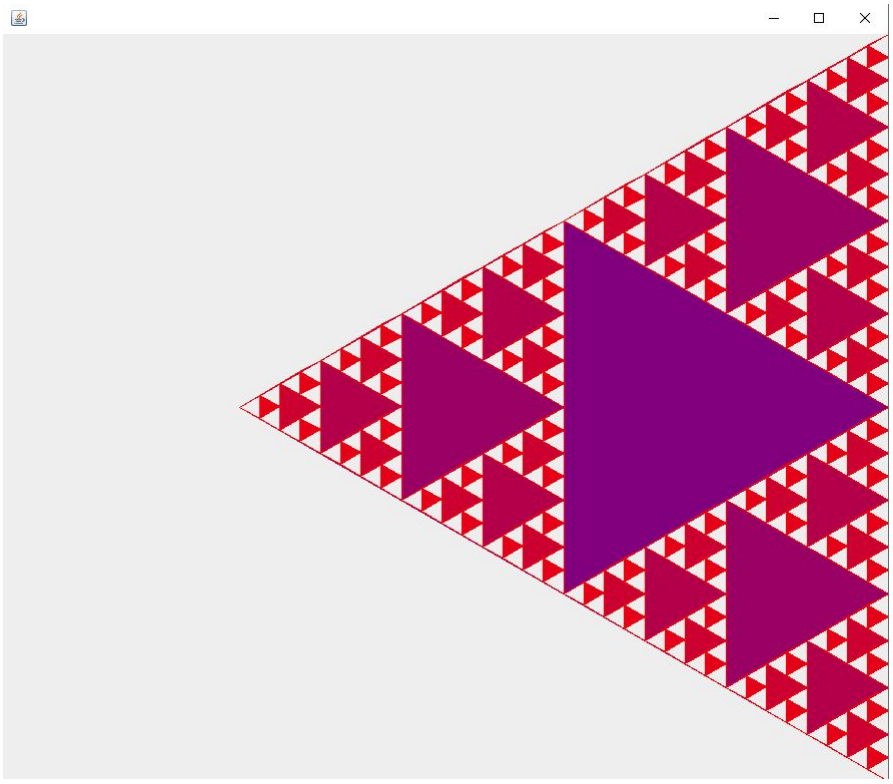
```
D.setSize(getParent().getSize());
height = D.height;
width = D.width;
```

The way it works is that every time the Frame gets resized, the paint() method gets called again, and with the calling of setSize() "D" gets updated to the current size of the frame, which in turns updates the values of the fields height and width.

Now once we execute the program, we get:



And if we resize it, so that the height is smaller than the width, we get:



While talking to Juri about our resizing problem, we mentioned the issues we had with the coloring of the triangle borders. He showed us his code but using it, we couldn't figure out where the problem lied. While browsing through his code, we saw his color randomiser and although ours worked perfectly fine, it could only do so for up to 10 recursions, and even if our screen can't even display 10 recursions properly, we did think of it as an issue. Which is why, we asked Juri if we could implement it in our code and after getting his permission, we implemented it as a method called randomColors(). Which we then called in the paint() method.

```
public void randomColors()
{
    Random rand = new Random();
    float r;
    float g;
    float b;
    for(int i = 0; i<recursion;i++) {
          r = rand.nextFloat();
          g = rand.nextFloat();
          b = rand.nextFloat();
          colors[i] = new Color(r,g,b);
    }
}
```

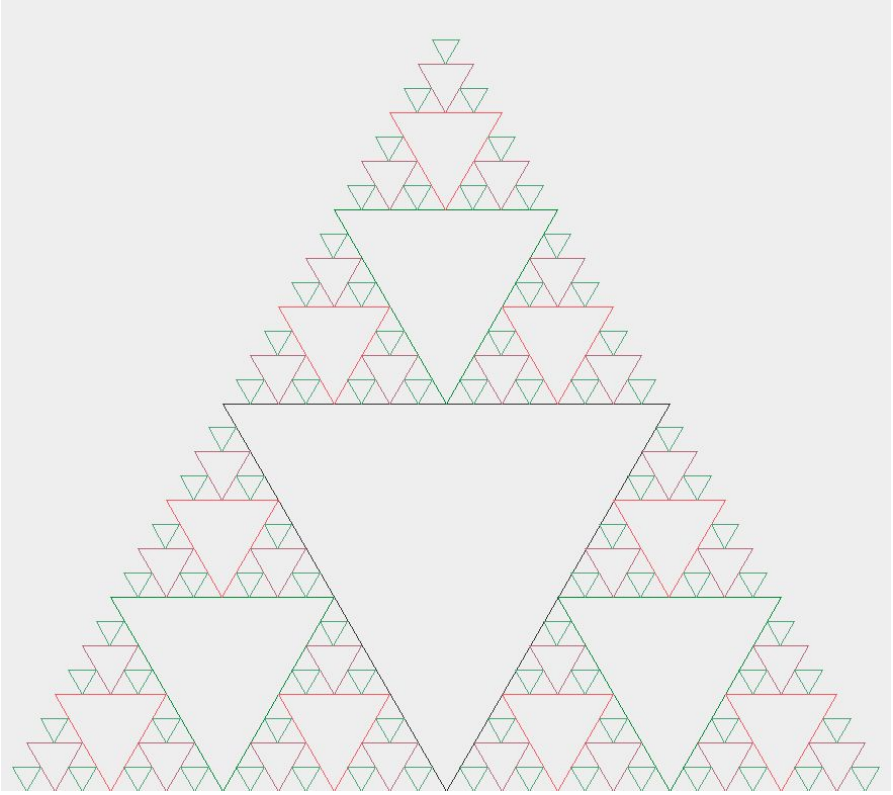
```
public void paint(Graphics g) {

    randomColors();
```

We then used the colors[] array with "recursion" as index, as parameters for the setColor() method in the drawTriangle() method.

```
g.setColor(colors[recursion]);
```

Now the only thing left to do, was to find a way to get the colored borders.

After a while, we managed to get the smaller triangles into different colors but at the same time we removed the big one, which looked like this:



We did this by moving the drawPolygon() from the if statement to the else statement and by changing its parameters to the arrays containing the coordinates from the midpoints.

```
int xxCoords[] = {midP12.getX(), midP13.getX(), midP23.getX()};
int yyCoords[] = {midP12.getY(), midP13.getY(), midP23.getY()};
//g.fillPolygon(xxCoords, yyCoords, 3);
g.drawPolygon(xxCoords, yyCoords, 3);
```

And after some more testing, we found a way of getting the big triangle back, without messing with the colors, which is to move the 2 arrays and the calling of the drawPolygon() method from the if statement, that we had moved for the 2nd exercise, back to the paint() method.

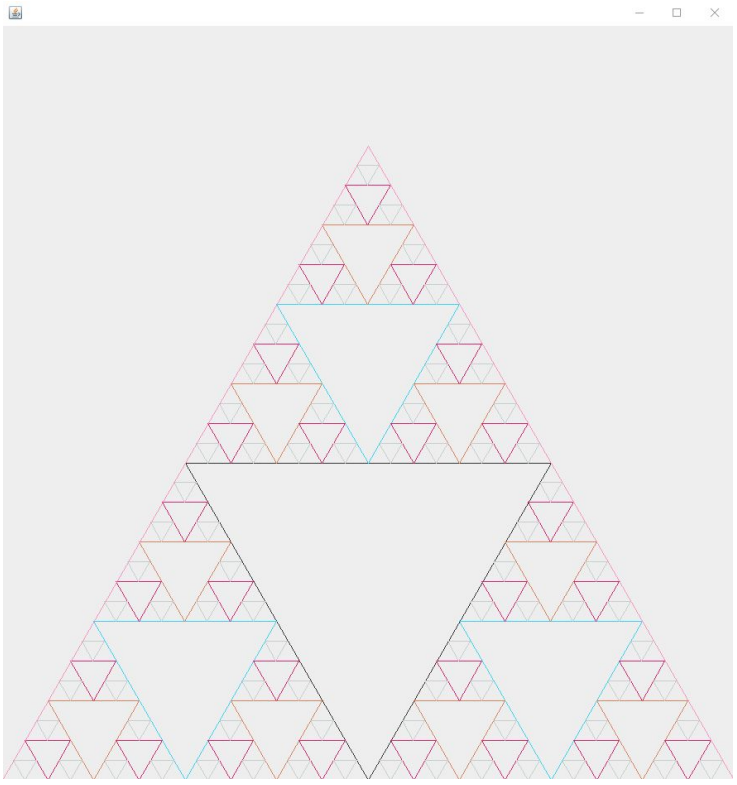
```
 int xCoords[] = {p1.getX(), p2.getX(), p3.getX()};
 int yCoords[] = {p1.getY(), p2.getY(), p3.getY()};

Graphics2D g2 = (Graphics2D) g;
drawTriangle(g2, p1, p2, p3, recursion);
g.drawPolygon(xCoords, yCoords, 3);
```

At the same time we changed the if-else statement into an if statement, since the if statement didn't serve any purpose anymore. The new if statement contains the same things as the previous else-statement, and its condition is

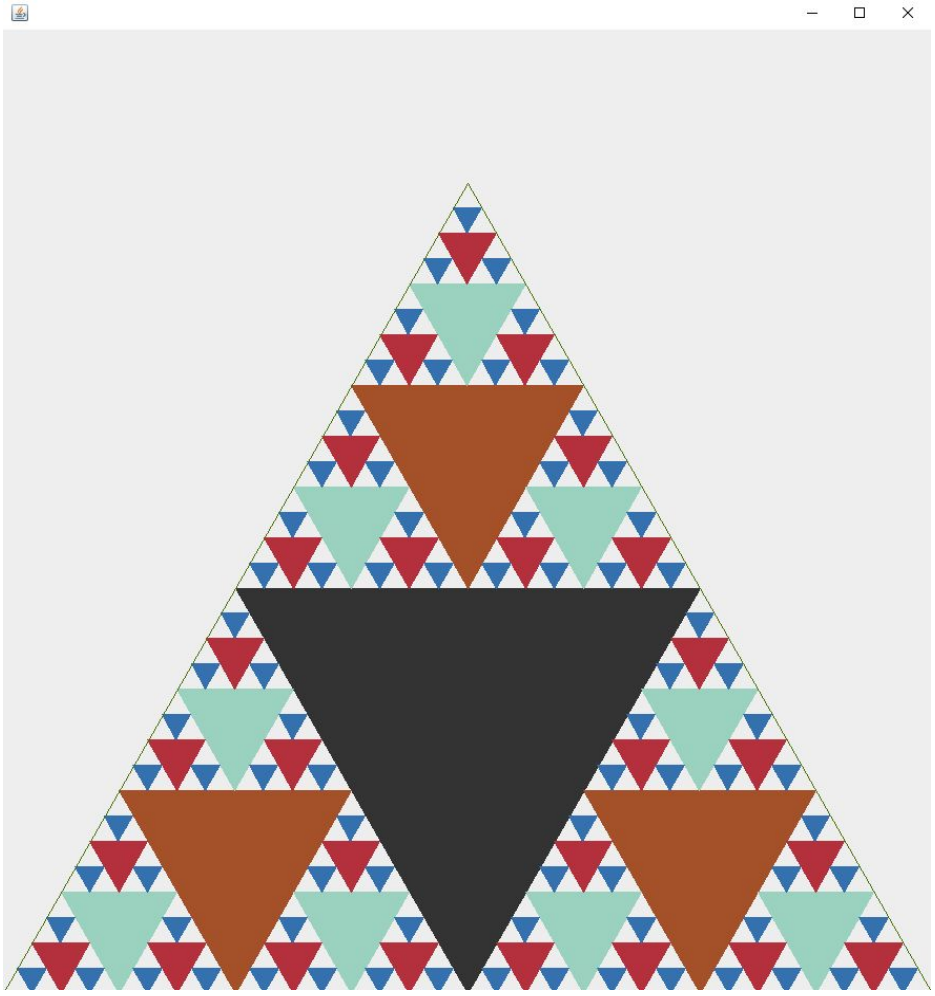
that recursion is not equal to 0.

```
if (recursion != 0) {

    int xxCoords[] = {midP12.getX(), midP13.getX(), midP23.getX()};
    int yyCoords[] = {midP12.getY(), midP13.getY(), midP23.getY()};
    g.fillPolygon(xxCoords, yyCoords, 3);
    g.drawPolygon(xxCoords, yyCoords, 3);

    drawTriangle(g, p1, midP12, midP13, recursion-1);
    drawTriangle(g, midP12, p2, midP23, recursion-1);
    drawTriangle(g, midP13, midP23, p3, recursion-1);

}
```

Now once we execute the program, we get:



And now, if we add the fillPolygon() back into the code, we get:



As we can see, it finally works as intended.

Reflections

Bartholomäus:

Since it was the first time that I had worked with recursion in programming, it had its challenges although not a lot. But in general, it was still fun to work on the recursive triangles.

Lars:

Recursion was a term that i knew but never worked with before. I was not sure how to create a formula to recreate triangles. Bartholomäus found out how to create it and even tho i understood it i don't believe i would have figured it out on my own. The other tasks to this project were pretty similar to a tutorial on oracle i worked on a couple years ago.

Bartholomäus Berresheim 568624
Lars Vorwerg 571683

Code

```
package ubung_9;

import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Color;
import java.awt.Dimension;
import java.util.Random;
import javax.swing.*;

public class Triangle extends JPanel {

	private static int width = 900;
	private static int height = 950;
	private static int h;
	private Dimension D = new Dimension(width, height);

	private static int recursion = 5;
	private static Color[] colors = new Color[recursion+1];

	  public static void main(String[] args) {

		JFrame frame = new JFrame();
		frame.getContentPane().add(new Triangle());
		frame.setSize(width,height);
		frame.setVisible(true);
		frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

	    randomColors();
	  }

	    public void paint(Graphics g) {

	      Point p1;
			Point p2;
			Point p3;

	      D.setSize(getParent().getSize());
			height = D.height;
			width = D.width;

	      if (height >= width) {
			p1 = new Point (0,height);
			p2 = new Point (width,height);
```

```
                h = (int) (height-(width/2*Math.sqrt(3)));
                p3 = new Point (width/2,h);
            }else {
                p1 = new Point (width,0);
                p2 = new Point (width,height);
                  h = (int) (width-(height/2*Math.sqrt(3)));
                  p3 = new Point (h,height/2);
            }
                 int xCoords[] = {p1.getX(), p2.getX(), p3.getX()};
                 int yCoords[] = {p1.getY(), p2.getY(), p3.getY()};

                Graphics2D g2 = (Graphics2D) g;
                drawTriangle(g2, p1, p2, p3, recursion);
                g.drawPolygon(xCoords, yCoords, 3);
          }

        public void drawTriangle(Graphics2D g, Point p1, Point p2, Point
p3, int recursion) {

                Color c = new Color(255-recursion*25,0,recursion*25);
                g.setColor(colors[recursion]);

                Point midP12 = getMidpoint (p1, p2);
                Point midP13 = getMidpoint (p1, p3);
                Point midP23 = getMidpoint (p2, p3);

             if (recursion != 0) {

                int xxCoords[] = {midP12.getX(), midP13.getX(),
midP23.getX()};
                int yyCoords[] = {midP12.getY(), midP13.getY(),
midP23.getY()};
                g.fillPolygon(xxCoords, yyCoords, 3);
                g.drawPolygon(xxCoords, yyCoords, 3);

                drawTriangle(g, p1, midP12, midP13, recursion-1);
                drawTriangle(g, midP12, p2, midP23, recursion-1);
                drawTriangle(g, midP13, midP23, p3, recursion-1);


          }

        }

        public Point getMidpoint(Point p1, Point p2) {
```

```
            Point p3 = new Point((p1.getX()+ p2.getX())/2,(p1.getY()+
p2.getY())/2);
            return p3;

        }

        public static void randomColors()
        {
              Random rand = new Random();
              float r;
              float g;
              float b;
              for(int i = 0; i<recursion;i++) {
                        r = rand.nextFloat();
                        g = rand.nextFloat();
                        b = rand.nextFloat();
                        colors[i] = new Color(r,g,b);
              }
        }
}
package ubung_9;

public class Point {

      private int x;
      private int y;

      Point(int x, int y){
            this.x = x;
            this.y = y;
      }
      public int getX() {
            return x;
      }
      public int getY() {
            return y;
      }
}
```