Lab Report

EXERCISE 9: RECURSIVE TRIANGLES

Name		Titel des Kurses	Datum
Juri Wiechmann	571085	Prof. Dr. Weber-Wulff Info 2 Group 2	04.01.2020
Bruno Dinis	570637		

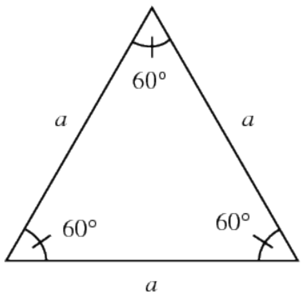
Index

Introduction

Pre-lab

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?

An equilateral triangle is a triangle which has in every corner the same angle of 60°. All edges have the same size.



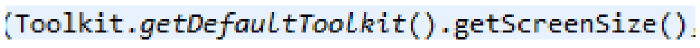
2. What is the mathematical formula for finding the midpoint of a line segment that connects two Points?

The midpoint of a line segment from P1(x1,y1) to P2(x2,y2) is

$$\left(\frac{x_1+x_2}{2}\ ,\ \frac{y_1+y_2}{2}\right)$$

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?

Normally you just take a look at your Graphic Driver. But you can get these too with Java. We used the Toolkit class.



```
Toolkit.getDefaultToolkit().getScreenSize()
```

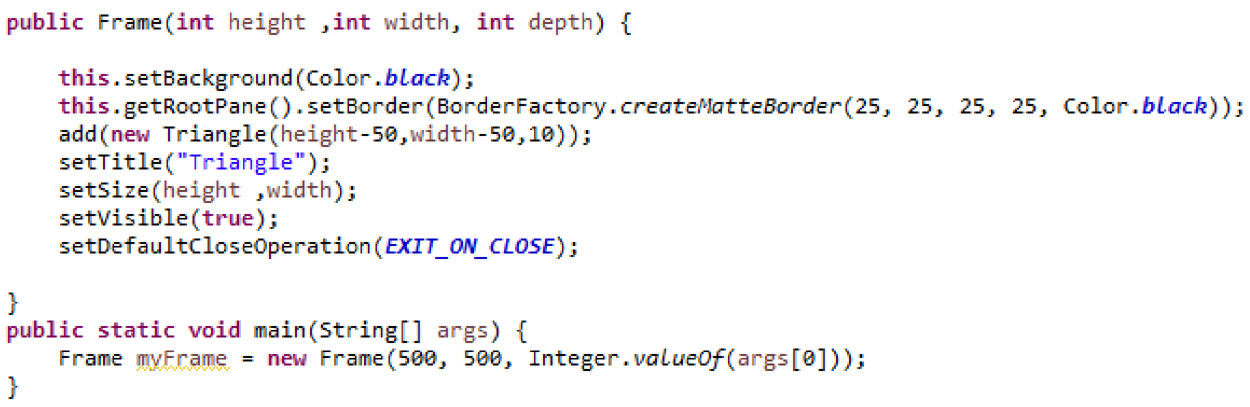
4. Briefly describe what a **Sierpinski Triangle**(S1) is.

The Sierpiksi triangle is a fractal with the overall shape of an equilateral triangle, subdivided recursively into smaller equilateral triangles(S1).

Assignments

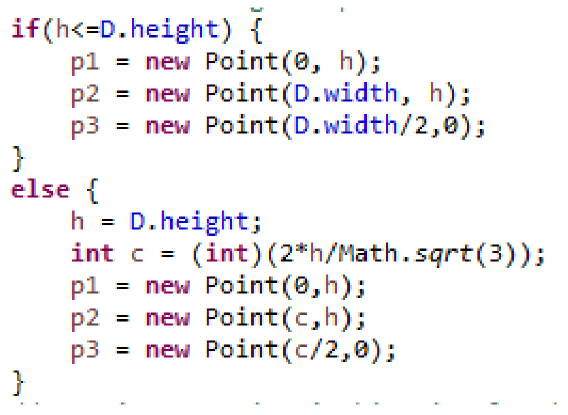
1. First set up a Window that can handle drawing. Can you get the Window to draw an equilateral triangle? What is the largest one you can get on the screen?

We decided to create a JFrame which contains our Triangle-class on which we want to draw. The Triangle-class is extended by a JPanel.



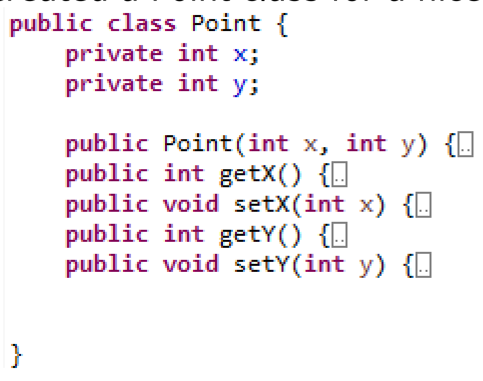
```
public Frame(int height ,int width, int depth) {

    this.setBackground(Color.black);
    this.getRootPane().setBorder(BorderFactory.createMatteBorder(25, 25, 25, 25, Color.black));
    add(new Triangle(height-50,width-50,10));
    setTitle("Triangle");
    setSize(height ,width);
    setVisible(true);
    setDefaultCloseOperation(EXIT_ON_CLOSE);

}
public static void main(String[] args) {
    Frame myFrame = new Frame(500, 500, Integer.valueOf(args[0]));
}
```

We want to have a little border because it looks better. Our Triangle constructor needs the size and the depth. We will later explain the purpose of the depth. We save our size in the Dimension-datatype. We need now 3 points to draw our equilateral triangle. The calculation of these points have 2 different cases. We get a rectangle as ground and now we want to draw a Triangle in it. We also want the maximum size of our Triangle that fits in this rectangle. In our first case the width is our limitation. In the second one the height is our limitation.



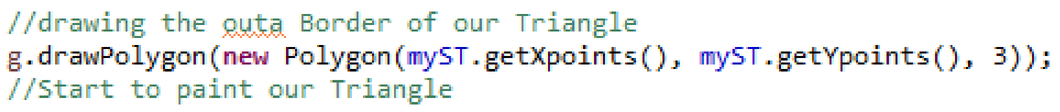
```
if(h<=D.height) {
    p1 = new Point(0, h);
    p2 = new Point(D.width, h);
    p3 = new Point(D.width/2,0);
}
else {
    h = D.height;
    int c = (int)(2*h/Math.sqrt(3));
    p1 = new Point(0,h);
    p2 = new Point(c,h);
    p3 = new Point(c/2,0);
}
```

We also created a Point class for a nicer overview.



```
public class Point {
    private int x;
    private int y;

    public Point(int x, int y) {⊡
    public int getX() {⊡
    public void setX(int x) {⊡
    public int getY() {⊡
    public void setY(int y) {⊡
}
```

If we want to draw we use the paintComponent(Graphics g) method from the JPanel. The Graphics-class gives us the possibility to draw a Polygon, so also a Triangle which looks like this:



```
//drawing the outa Border of our Triangle
g.drawPolygon(new Polygon(myST.getXpoints(), myST.getYpoints(), 3));
//Start to paint our Triangle
```

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?

Our Idea is to create a complex Datatype specific for the Sierpinski Triangle and independent from the methods. We got 7 fields. 3 of them to save the points, 1 for the depth and the last 3 to contain the 3 lower equilateral triangles. The amount of levels our Sierpinski Triangle has is determined by the depth (Different words, same meaning). Our constructor asks for the first 3 points and the depth. After we copied them to the fields we create the lower Triangles. We reduce the depth by one. Calculate all 3 new midpoints and create 3 new objects with the same type. It is important to not get confused with the points that we now put in the 3 new objects.

```java
public class Sierpinski_Triangle {

    private Point p1;
    private Point p2;
    private Point p3;
    public Point getP1() {□
    public Point getP2() {□
    public Point getP3() {□
    public int[] getXpoints(){□
    public int[] getYpoints(){□

    //Safeing the lower Triangles.
    private Sierpinski_Triangle st1;
    private Sierpinski_Triangle st2;
    private Sierpinski_Triangle st3;

    public Sierpinski_Triangle getSt1() {□
    public Sierpinski_Triangle getSt2() {□
    public Sierpinski_Triangle getSt3() {□

    private int depth;
    public int getDepth() {
        return depth;
    }

    public Sierpinski_Triangle(int depth, Point p1, Point p2, Point p3) {
        this.p1 = p1;
        this.p2 = p2;
        this.p3 = p3;
        this.depth = depth;
        if(depth != 0) {
            depth--;
            Point midP1 = UM.getMidpoint(p1, p2);
            Point midP2 = UM.getMidpoint(p2, p3);
            Point midP3 = UM.getMidpoint(p1, p3);
            st1 = new Sierpinski_Triangle(depth, p1, midP1, midP3);
            st2 = new Sierpinski_Triangle(depth, midP1, p2, midP2);
            st3 = new Sierpinski_Triangle(depth, midP3, midP2, p3);
        }
    }
}
```

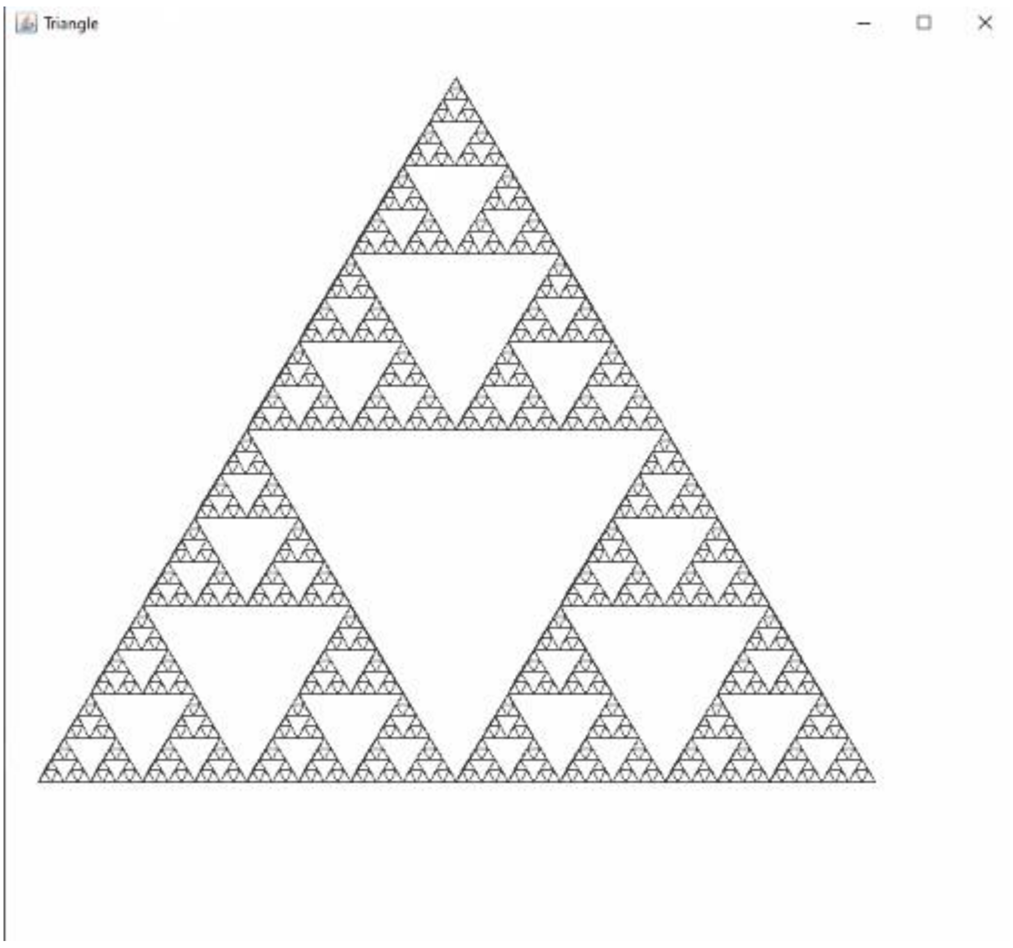
Now we create one of these in our Triangle-class with the calculated points. To get all the sub triangles we create a recursive method. This method calculates 3 new midpoints to draw the inner Triangle

```java
//Drawing Lines are the connection of all Midpoints of our edges.
int[]xpoints = {
        UM.getMidpoint(myST.getP1(), myST.getP2()).getX(),
        UM.getMidpoint(myST.getP2(), myST.getP3()).getX(),
        UM.getMidpoint(myST.getP1(), myST.getP3()).getX(),
};
int[]ypoints = {
        UM.getMidpoint(myST.getP1(), myST.getP2()).getY(),
        UM.getMidpoint(myST.getP2(), myST.getP3()).getY(),
        UM.getMidpoint(myST.getP1(), myST.getP3()).getY(),
};
g.drawPolygon(new Polygon(xpoints, ypoints, 3));
```

The recursive part comes now. We also want to draw the sub-triangles. To do so we need to check 2 conditions. Are there any sub-triangles and is it worth to draw them. To check if there are any-triangles one level lower we just check if depth is greater than 0. To check if it is worth to draw them we said that every Triangle which is smaller than 3 isn't worth. So we check the distance between 2 points.

```
//if there are more and
//if its worth to paint them(Triangle is bigger than 3 Pixels)
//We call this function again 3 Time for the next 3 Triangle with one size lvl lower.
if(myST.getDepth()>0 && myST.getP2().getX()-myST.getP1().getX()>3) {
    paintTriangles(g, myST.getSt1());
    paintTriangles(g, myST.getSt2());
    paintTriangles(g, myST.getSt3());
}
```

The result looks like this:



Older picture, later we changed the background to dark mode because darkmode is WAY cooler.

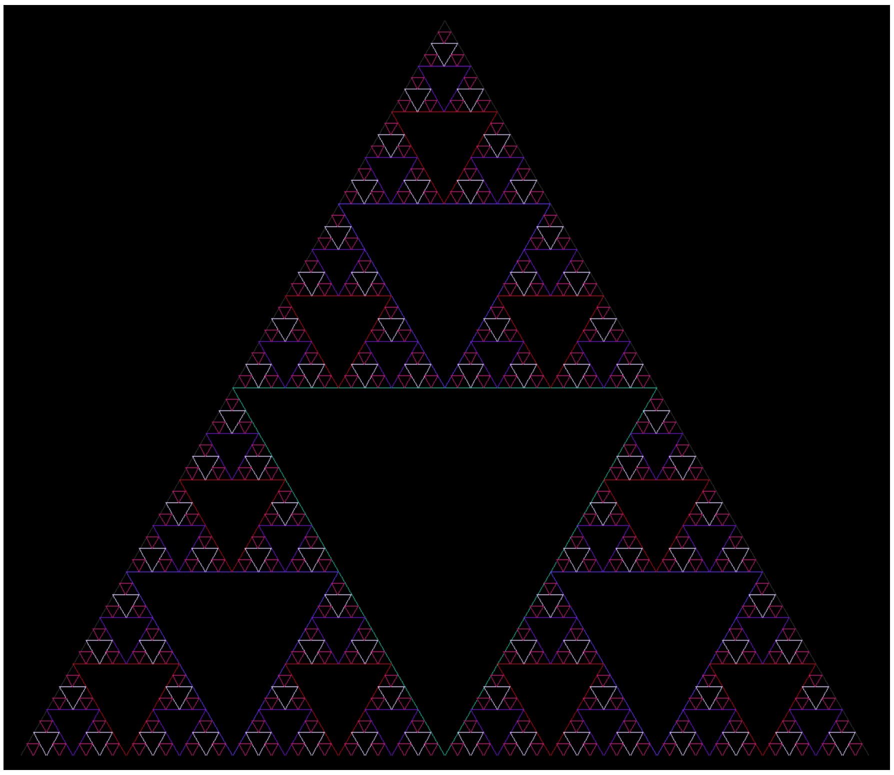
3. Expand your triangle drawing algorithm to draw in a specific color. Choose a different color for every level of the algorithm.

We decided to create an array of random colors. The depth now works as an index so every Triangle with the same size have the same color.

```java
public Triangle(int height, int width, int depth) {
    D = new Dimension(width, height);
    this.depth = depth;
    colors = new Color[depth+2];
    Random rand = new Random();
    float r;
    float g;
    float b;
    for(int i = 0; i<colors.length;i++) {
        r = rand.nextFloat();
        g = rand.nextFloat();
        b = rand.nextFloat();
        colors[i] = new Color(r,g,b);
    }
    setBackground(Color.black);
}
```

Now we simply need to change the color that corresponds to the depth and we are finished:

```java
g.setColor(colors[myST.getDepth()]);
```

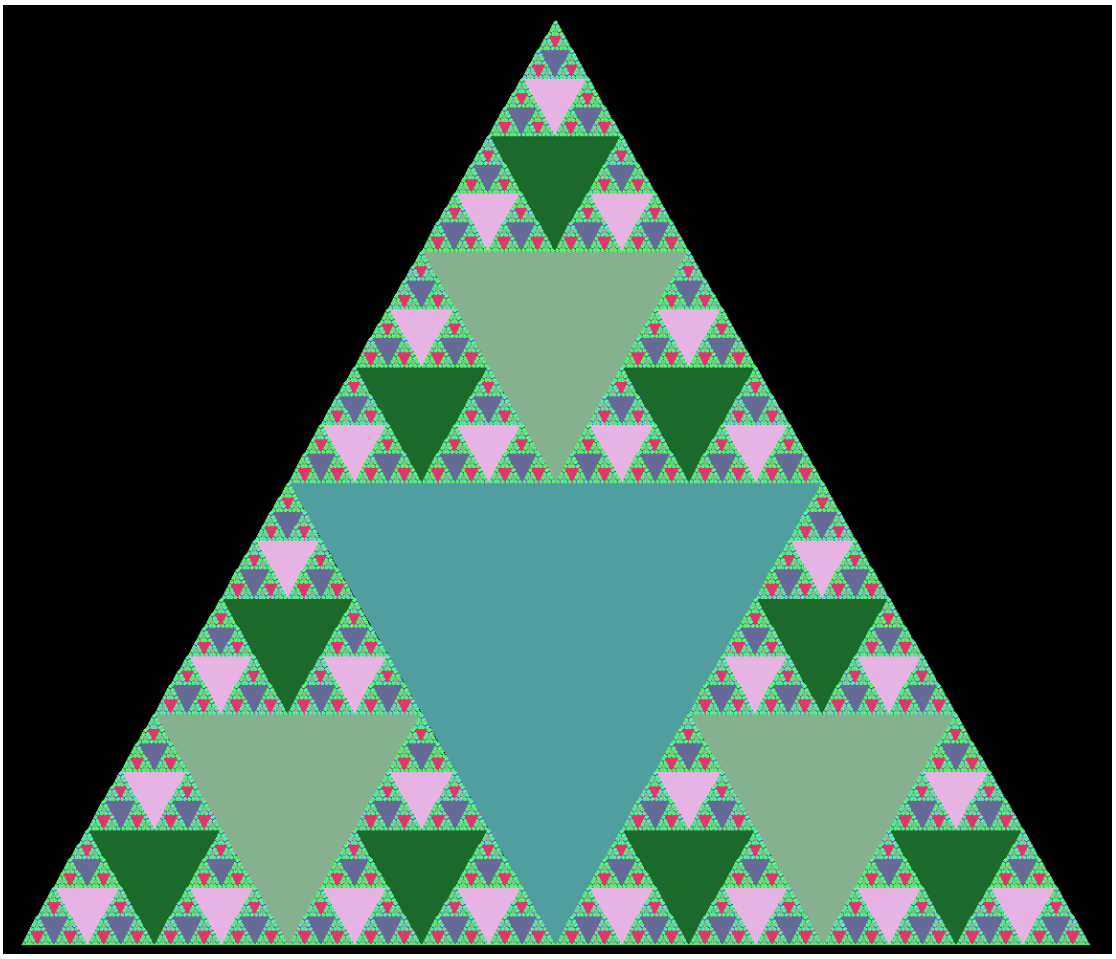


4. Fill the middle triangle on each step with an appropriate color. Choose the size of the first triangle depending on what size the window is. Redraw the triangle when the window is resized.

For this we just need to add a line:

```
g.fillPolygon(new Polygon(xpoints, ypoints, 3));
```

And we are finished. Because it’s random every time you get a nice new Triangle:



Reflections

Bruno

Overall, this lab was quite enjoyable since it was a good revision of different subjects we had learned before like drawing in JPanel (using x and y coordinates) and using recursions. The best part about this lab was seeing the colorful triangle that we get as a result.

Juri

To be honest this was for Christmas but we did it in the last Weekend. It was still pretty but we hadn't the time to do some extra stuff. Plans like an infinite zoom or other fractals dropped out. I hope had a nice Christmas and a happy New Year.

Source:

(S1): https://en.wikipedia.org/wiki/Sierpi%C5%84ski_triangle

Appendix:

```
package ubung9;

import java.awt.Color;
import java.awt.Component;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.GridBagLayout;
import java.awt.Toolkit;
import java.awt.event.ComponentEvent;
import java.awt.event.ComponentListener;

import javax.swing.BorderFactory;
import javax.swing.JFrame;
import javax.swing.JWindow;

public class Frame extends JFrame{

      public Frame(int height ,int width, int depth) {

            this.setBackground(Color.black);

      this.getRootPane().setBorder(BorderFactory.createMatteBorder(
25, 25, 25, 25, Color.black));
            add(new Triangle(height-50,width-50,depth));
            setTitle("Triangle");
```

```
		setSize(height ,width);
		setVisible(true);
		setDefaultCloseOperation(EXIT_ON_CLOSE);

	}
	public static void main(String[] args) {
		System.out.println(args[0]);
		Frame myFrame = new Frame(500, 500, 
Integer.valueOf(args[0]));
	}
}
package ubung9;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Polygon;
import java.util.Random;

import javax.swing.JPanel;


public class Triangle extends JPanel {

	private Sierpinski_Triangle myST;
	private Dimension D;
	//every different size of Triangle got a random Color.
	private Color[] colors;
	//Amount of iterations
	private int depth;
	/*
	 * A triangle drawing panel can be called with a Dimension or 
by the value directly.
	 */
	public Triangle(Dimension dimension, int depth) {
		Triangle myTriangle = new 
Triangle(dimension.height,dimension.width,depth);
	}
	public Triangle(int height, int width, int depth) {
		D = new Dimension(width, height);
		this.depth = depth;
		colors = new Color[depth+2];
		Random rand = new Random();
		float r;
		float g;
		float b;
		for(int i = 0; i<colors.length;i++) {
			r = rand.nextFloat();
			g = rand.nextFloat();
			b = rand.nextFloat();
			colors[i] = new Color(r,g,b);
		}
		setBackground(Color.black);
	}
	@Override

	public void paintComponent(Graphics g) {
		//Ask again for the Size everytime to update the 
Triangle.
		D.setSize(getParent().getSize());

		//calculating the hight of the Triangle.
		int height = D.height;
		int h = (int) Math.round(D.width/2*Math.sqrt(3));
		h = height -(D.height-h);

		//calculating the 3 corner points.
		Point p1;
		Point p2;
		Point p3;
		//checking if our width or height of our Panel is 
limiting our Triangle
		//and calculating the points in corresponds to that.
		if(h<=D.height) {
			p1 = new Point(0, h);
			p2 = new Point(D.width, h);
			p3 = new Point(D.width/2,0);
		}
		else {
			h = D.height;
			int c = (int)(2*h/Math.sqrt(3));
			p1 = new Point(0,h);
			p2 = new Point(c,h);
			p3 = new Point(c/2,0);
		}
		//creating new Sierpinski Triangle with the points.
		myST = new Sierpinski_Triangle(depth, p1, p2, p3);

		//drawing the outa Border of our Triangle
		g.drawPolygon(new Polygon(myST.getXpoints(), 
myST.getYpoints(), 3));
		//Start to paint our Triangle
		g.setColor(colors[myST.getDepth()+1]);
		paintTriangles(g, myST);
	}
	public void paintTriangles(Graphics g, Sierpinski_Triangle 
myST) {
		g.setColor(colors[myST.getDepth()]);
		//Drawing Lines are the connection of all Midpoints of 
our edges.
		int[]xpoints = {
				UM.getMidpoint(myST.getP1(), 
myST.getP2()).getX(),
                        UM.getMidpoint(myST.getP2(), 
myST.getP3()).getX(),
                        UM.getMidpoint(myST.getP1(), 
myST.getP3()).getX(),
            };
            int[]ypoints = {
                        UM.getMidpoint(myST.getP1(), 
myST.getP2()).getY(),
                        UM.getMidpoint(myST.getP2(), 
myST.getP3()).getY(),
                        UM.getMidpoint(myST.getP1(), 
myST.getP3()).getY(),
            };
            g.drawPolygon(new Polygon(xpoints, ypoints, 3));
            g.fillPolygon(new Polygon(xpoints, ypoints, 3));

            //if there are more and
            //if its worth to paint them(Triangle is bigger than 3 
Pixels)
            //We call this function again 3 Time for the next 3 
Triangle with one size lvl lower.
            if(myST.getDepth()>0 && myST.getP2().getX()-
myST.getP1().getX()>3) {
                  paintTriangles(g, myST.getSt1());
                  paintTriangles(g, myST.getSt2());
                  paintTriangles(g, myST.getSt3());
            }
      }
}
package ubung9;


public class Sierpinski_Triangle {

      private Point p1;
      private Point p2;
      private Point p3;
      public Point getP1() {
            return p1;
      }
      public Point getP2() {
            return p2;
      }
      public Point getP3() {
            return p3;
      }
      public int[] getXpoints(){
            int[]xpoints = {
                        getP1().getX(),
                        getP2().getX(),
                        getP3().getX()
            };
```

```
			return xpoints;
		}
		public int[] getYpoints(){
			int[]ypoints = {
					getP1().getY(),
					getP2().getY(),
					getP3().getY()
			};
			return ypoints;
		}

		//Safeing the lower Triangles.
		private Sierpinski_Triangle st1;
		private Sierpinski_Triangle st2;
		private Sierpinski_Triangle st3;

		public Sierpinski_Triangle getSt1() {
			return st1;
		}
		public Sierpinski_Triangle getSt2() {
			return st2;
		}
		public Sierpinski_Triangle getSt3() {
			return st3;
		}

		private int depth;
		public int getDepth() {
			return depth;
		}

		public Sierpinski_Triangle(int depth, Point p1, Point p2, 
Point p3) {
			this.p1 = p1;
			this.p2 = p2;
			this.p3 = p3;
			this.depth = depth;
			if(depth != 0) {
				depth--;
				Point midP1 = UM.getMidpoint(p1, p2);
				Point midP2 = UM.getMidpoint(p2, p3);
				Point midP3 = UM.getMidpoint(p1, p3);
				st1 = new Sierpinski_Triangle(depth, p1, midP1, 
midP3);
				st2 = new Sierpinski_Triangle(depth, midP1, p2, 
midP2);
				st3 = new Sierpinski_Triangle(depth, midP3, 
midP2, p3);
			}
		}
}
package ubung9;
```

```
public class Point {
	private int x;
	private int y;

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


}
package ubung9;


public interface UM {

	public static String deleteSpaces(String s) {

		return s.replace("\\s+", "");
	}

	public static boolean StringisNumber(String number) {

		if(number.startsWith("."))
			return false;

		char[] cNumber = number.toCharArray();

		int i = 0;
		try {
			if(cNumber[0] == '-' &&
CharisNumberHEX(cNumber[1]))
				i = 1;
		}
		catch (Exception e) {}

		for (; i < cNumber.length ; i++) {
			if(!(CharisNumberHEX(cNumber[i]) || cNumber[i] ==
'.'))
```

```
                        return false;
                }
                return true;
        }
        public static boolean CharisNumberHEX(char number) {
                if( number > 47 && number < 58 ||
                        number > 64 && number < 71)
                        return true;
                else
                        return false;
        }
        public static String setSpacesEachToken(String S) {
                S = deleteSpaces(S);
                char [] S_Array = S.toCharArray();

                //If the input String has 2 dots in a row its wrong
Tested here.
                int count = 0;
                for(char E : S_Array) {
                        if(E == '.')
                                count++;
                        else
                                count = 0;
                        if(count >1)
                                return null;
                }

                String token = "";
                S = "";
                for(int i = 0; i< S_Array.length;i++) {
                        token = "";

                        //negativ numbers
                        if(   S_Array[i] == '-') {
                                token += S_Array[i];

                                try {
                                        //if its an neg followed by a number
                                        if(CharisNumberHEX(S_Array[i+1])){
                                                //If - is the first digit:
                                                if(i==0) {

        while(CharisNumberHEX(S_Array[i+1]) || S_Array[i+1] == '.') {
                                                                i++;
                                                                token +=
S_Array[i];
                                                        }
                                                }
                                                else {
                                                        //if the - is not at the
first digit
```

```
       if(CharisOperator(S_Array[i-1])) {

       while(CharisNumberHEX(S_Array[i+1]) || S_Array[i+1] == '.') {
                                                     i++;
                                                     token +=
S_Array[i];
                                                 }
                                             }
                                         }
                                     }
                                     //if the neg is followed by an (
                                     else if(S_Array[i+1] == '(') {
                                         i++;
                                         token += S_Array[i];
                                     }
                                 }
                                 catch(Exception e) {}
                                 S += token + " ";
                                 continue;
                             }
                             //operatoren
                             if( i > 0  && CharisOperator(S_Array[i])) {
                                 token += S_Array[i];
                                 S += token + " ";
                                 continue;
                             }
                             //positive Zahlen
                             else if(CharisNumberHEX(S_Array[i])){
                                 token = Character.toString(S_Array[i]);
                                 try {
                                     while(CharisNumberHEX(S_Array[i+1])
|| S_Array[i+1] == '.') {
                                         i++;
                                         token += S_Array[i];
                                     }
                                 }
                                 catch(Exception e) {}
                                 S += token + " ";
                                 continue;
                             }
                             else {
                                 //If input is wrong.
                                 return null;
                             }
                         }


                         return S.stripTrailing();
                     }
                     public static boolean CharisToken(char c) {
                         if(CharisNumberHEX(c) || CharisOperator(c)) {
                             return true;
```

```
			}
			else
				return false;
		}
		public static boolean CharisOperator(char c) {
			if( c == '+' ||
				c == '-' ||
				c == 'x' ||
				c == '*' ||
				c == '/' ||
				c == '^' ||
				c == '(' ||
				c == ')' ||
				c == '=')
				return true;
			else
				return false;
		}

		public static String getSep(String input) {
			//if you want too really seaching for an sepperator u
	need a more complicated way, this is the bugded version
			if(input.contains("/"))
				return  "/";
			else if(input.contains("-"))
				return  "-";
			else if(input.contains("."))
				return  ".";
			else if(input.contains("_"))
				return  "_";
			return null;
		}
		public static int twoPointsDistance(int x1, int y1, int x2,
	int y2) {

			return (int) Math.sqrt((y2 - y1) * (y2 - y1) + (x2 -
	x1) * (x2 - x1));
		}
		public static Point getMidpoint(Point p1, Point p2) {

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