Sahar K Hussain
CAP3027
Section 1925
October 28rd, 2015
HW10

The work submitted is my own and the honor code was neither bent nor broken.

Sahar K Hussain

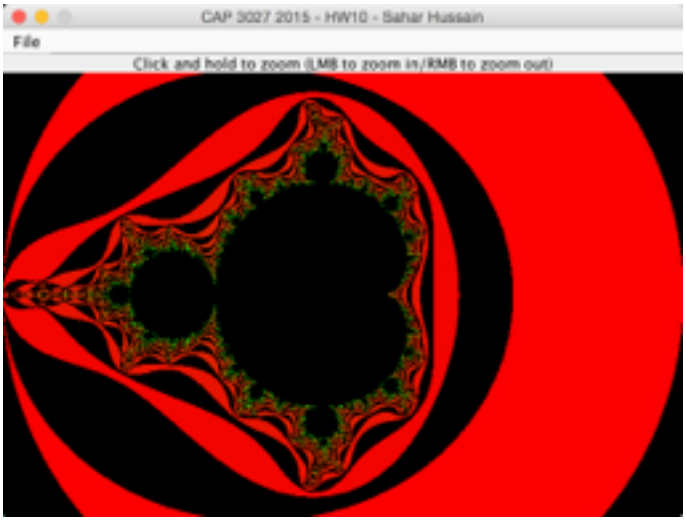
The easiest part of the homework was building a little bit off of my last homework. The hardest part was actually figuring out how to compute the zooming in and zooming out parts of the function I believe the assignment's educational objective was to help us practice using new techniques of java. It was gratifying after figuring out how to compute the image because I changed the style to dragon tiger and wanted to push myself more for this homework

Sahar K Hussain
CAP3027
Section 1925
October 28, 2015
HW10

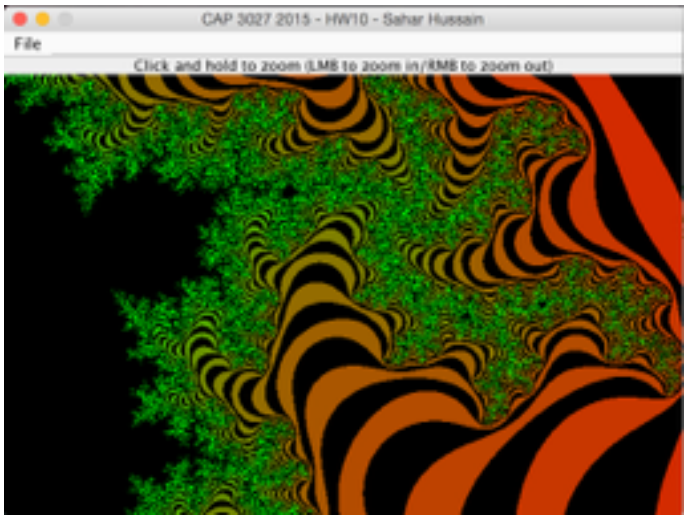
Standard Program Deliverables:
1) Yes, the program compiles without errors.
2) Yes, the program compiles without warnings.
3) Yes, the program runs without crashing.
4) I tested the program on my personal laptop and utilized xCode for inputting the code and the terminal for compiling and running the program.
5) The program does meet assignment's specifications - N/A
6) No known and suspected bugs.
7) Yes, the program runs correctly.

Screenshots:

Unzoomed mandelbrot

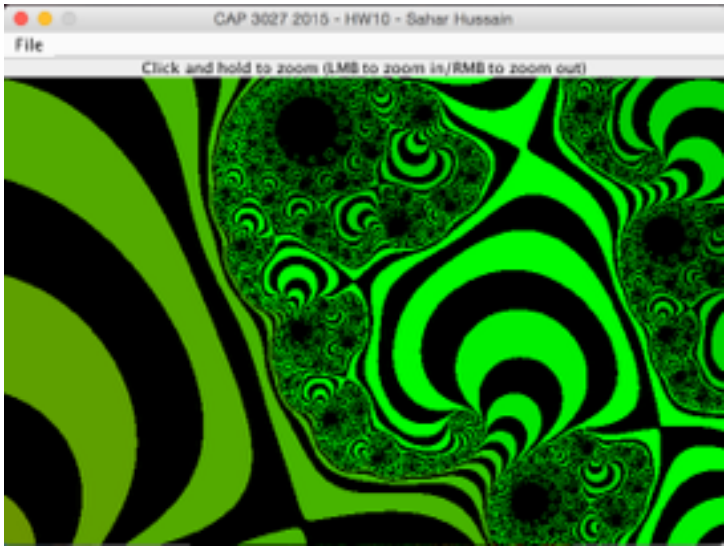


Zoomed Mandelbrot



Unzoomed Julia

Zoomed Julia



Source Code:

```
//DisplayImage.java

//Allows a user to select and display images

//illustrates how to create a JFrame with a menubar,

//define ActionListeners,

//use a JFileChooser,

//open and display an image inside a JScrollPane


//by Dave Small

//HW10 Modification to original HW00 code and work by Sahar KH


/**For this assignment, I shall be implementing one of the Mandelbrot/Julia Set Graphics Techniques

 as described in Lecture: Mandelbrot & Julia Sets but this time be implementing a Mandelbrot set and Julia
```

```
 set explorer with animated (un)zooming.
 **/

// Import the libraries

import java.awt.Color;
import java.awt.Graphics2D;
import java.awt.List;
import java.awt.Rectangle;
import java.awt.Shape;
import java.awt.geom.AffineTransform;
import java.awt.geom.Line2D;
import java.awt.geom.*;
import java.awt.geom.Point2D;
import java.awt.*;
import java.lang.Math;
import java.lang.Character;
import java.util.Stack;
import java.util.Scanner;
import java.awt.event.*;
import java.awt.image.*;
import java.io.*;
import javax.imageio.*;
import java.awt.Graphics;
import java.lang.Math.*;
```

```
import java.lang.Math;
import javax.swing.*;
import javax.swing.JLabel;
import javax.swing.JMenuBar;
import javax.swing.JMenu;
import javax.swing.JMenuItem;
import java.util.ArrayList;
import java.io.IOException;
import java.io.FileReader;
import java.io.BufferedReader;
import java.util.Random;
import javax.swing.ImageIcon;
import javax.swing.JOptionPane;
import javax.swing.*;
import java.lang.Object;
import java.awt.Robot;


class DisplayImage
{
     // Has a fractal display panel 600 pixels wide and 450 pixels tall
    private static final int WIDTH = 600;

    private static final int HEIGHT = 450;
```

```
    // Our worker thread called by the EDT to run the program in a safe way
    public static void main(String[] args)
    {
        SwingUtilities.invokeLater(new Runnable()
                                {
            public void run()
            {
                createAndShowGUI();
            }
        });
    }
    private static void createAndShowGUI()
    {
        JFrame frame = new ImageFrame(WIDTH, HEIGHT);

        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        frame.setVisible(true); // frame.show(); was deprecated as of Java 1.5
    }
}

class ImageFrame extends JFrame
{
    // Initializes our variables
```

```
    private final int WIDTH;

    private final int HEIGHT;

    private BufferedImage image = null;

    private AreaSelectPanel panel;

    private JFileChooser chooser;

    private Graphics2D g2;

    private boolean isJulia = false, isZooming = false, isCentering = false, isCentered =
false, zoomingIn = false;

    private double firstReal, firstImage, firstRealFinal, firstImageFinal;

    private double secondReal, secondImage, secondRealFinal, secondImageFinal;

    private double mu_real, mu_i;

    private Timer zoomTimer;

    private final int MILLISECONDS_BETWEEN_FRAMES = 17;
```

```
private Robot robot;

private double dr, di, newCenterX, newCenterY;

private Point2D.Double point;

private JLabel label;

private JLabel imageLabel;

//  Constructor

public ImageFrame(int width, int height)
{
   try
   {
      robot = new Robot();

   }catch(AWTException e){}

   WIDTH = width;

   HEIGHT = height;

   // creates new graphics
```

```
image = new BufferedImage(width, height, BufferedImage.TYPE_INT_ARGB);

g2 = (Graphics2D) image.createGraphics();

//setup the frame's attributes

// The title-bar describes the assignment (CAP 3027 2015 - HWxx - your name)

this.setTitle("CAP 3027 2015 - HW10 - Sahar Hussain");

this.setSize(width, height);

try
{
   robot = new Robot();

}catch(AWTException e){}

//add a menu to the frame
addMenu();

chooser = new JFileChooser();

chooser.setCurrentDirectory(new File("."));
```

```
firstReal = -2.0;

firstImage = 1.5;

firstRealFinal = 2.0;

firstImageFinal = -1.5;

secondReal = -2.0;

secondImage = 1.5;

secondRealFinal = 2.0;

secondImageFinal = -1.5;

mu_real = 0.0;

mu_i = 0.0;

newCenterX = 300.0;

newCenterY = 225.0;

dr = Math.abs(firstRealFinal - firstReal);
```

```
di = Math.abs(firstImageFinal - firstImage);

point = new Point2D.Double(0.0, 0.0);

mandelbrotImageMaker(-2.0, 1.5, 2.0, -1.5);

zoomTimer = new Timer(MILLISECONDS_BETWEEN_FRAMES, new ActionListener()
                {
   public void actionPerformed(ActionEvent evt)
   {
      if(zoomingIn)
      {
         zoomTimer.stop();
         zoomIn();
         zoomTimer.restart();
      }
      else
      {
         zoomTimer.stop();
         zoomOut();
         zoomTimer.restart();
      }
   }
});
```

```
addMouseListener(new MouseAdapter()
            {
   public void mousePressed(MouseEvent event)
   {
      if(event.getButton() == event.BUTTON1)
      {
         zoomingIn = true; //When the left mouse button is clicked, zoom in
         //zoomingIn = false; //When the left mouse button is clicked, zoom out
      }
      else if(event.getButton() == event.BUTTON3)
      {
         zoomingIn = false; //When the right mouse button is clicked, zoom out
         //zoomingIn = true; //When the right mouse button is clicked, zoom in
      }
      isZooming = true;
      zoomTimer.start();
      centerImage((double)event.getX(), (double)event.getY() - 25);
   }
   public void mouseReleased(MouseEvent event)
   {
      zoomTimer.stop();
   }
});
```

```
        // Has a label: "Click and hold to zoom (LMB to zoom in/RMB to zoom out)"

        //label1 = new JLabel( "Click and hold to zoom (LMB to zoom in/RMB to zoom
out)" );

        //JFrame frame = new JFrame();

        //frame.getContentPane().add( label1, BorderLayout.SOUTH );

        //frame.getContentPane().add( panel, BorderLayout.CENTER );

        //frame.pack();

        //frame.setVisible( true);

        label = new JLabel("Click and hold to zoom (LMB to zoom in/RMB to zoom out)");
        label.setHorizontalAlignment(JLabel.CENTER);
        this.add(label, BorderLayout.NORTH);



        imageLabel = new JLabel(new ImageIcon(image));
        this.add(imageLabel);

        this.setResizable(false);
    }
```

```
    //--------------------------------- Methods that Implement the
Menu----------------------------------------//

    /**

     The File menu has the following selections

     Mandelbrot

     Julia

     Save image

     Exit

     **/

    private void addMenu()
    {
        //setup the frame's menu bar
        // === file menu

        JMenu fileMenu = new JMenu("File");

        // The JMenuItem that will load Mandelbrot image

        JMenuItem mandelBrotImage = new JMenuItem( "Mandelbrot" );
        mandelBrotImage.addActionListener(new ActionListener()
```

```
                {
public void actionPerformed(ActionEvent event)
{
   firstReal = -2.0;

   firstImage = 1.5;

   firstRealFinal = 2.0;

   firstImageFinal = -1.5;

   secondReal = -2.0;

   secondImage = 1.5;

   secondRealFinal = 2.0;

   secondImageFinal = -1.5;

   dr = Math.abs(firstRealFinal - firstReal);

   di = Math.abs(firstImageFinal - firstImage);

   mandelbrotImageMaker(-2.0, 1.5, 2.0, -1.5);
}
```

```
});

fileMenu.add(mandelBrotImage);

// The JMenuItem that will load the Julia image

JMenuItem juliaImage = new JMenuItem("Julia");
juliaImage.addActionListener(new ActionListener()
                    {
   public void actionPerformed(ActionEvent event)
   {
      isZooming = false;

      firstReal = -2.0;

      firstImage = 1.5;

      firstRealFinal = 2.0;

      firstImageFinal = -1.5;

      secondReal = -2.0;

      secondImage = 1.5;
```

```
        secondRealFinal = 2.0;

        secondImageFinal = -1.5;

        dr = Math.abs(firstRealFinal - firstReal);

        di = Math.abs(firstImageFinal - firstImage);

        juliasetImageMaker(-2.0, 1.5, 2.0, -1.5);
    }
});

fileMenu.add(juliaImage);

// The JMenuItem that will save the image

JMenuItem saveOurImage = new JMenuItem("Save image");
saveOurImage.addActionListener(new ActionListener()
                        {
    public void actionPerformed(ActionEvent event)
    {
        saveTheImage();
    }
});
fileMenu.add( saveOurImage );
```

```
        //Exit
        JMenuItem exitItem = new JMenuItem("Exit");
        exitItem.addActionListener(new ActionListener()
                                {
           public void actionPerformed(ActionEvent event)
           {
              System.exit(0);
           }
        });
        fileMenu.add(exitItem);

        //attach a menu to a menu bar

        JMenuBar menuBar = new JMenuBar();
        menuBar.add(fileMenu);
        this.setJMenuBar(menuBar);
    }

    //-------------------------------------------------------------------------------------------
----------//

    //---------------------------------- Methods that get inputs from the
user-----------------------------------------//

    // Prompts the user for the desired real number for Mu
```

```
    private double userMuReal() {

        double promptMuReal = 0;

        try {

            String prompt = JOptionPane.showInputDialog("Please input the desired real
number for Mu [-2.0, 2.0]");

            promptMuReal = Double.parseDouble(prompt);

        }

        catch(NumberFormatException e) {

            JOptionPane.showMessageDialog(this, "Error, please input the desired real
number for Mu [-2.0, 2.0]");

            String prompt = JOptionPane.showInputDialog("Please enter the desired real
number for Mu");

            promptMuReal = Double.parseDouble(prompt);

        }

        return promptMuReal;

    }


    // Prompts the user for the desired imaginary number for Mu

    private double userMuImaginary() {

        double promptMuImaginary = 0;

        try {

            String prompt = JOptionPane.showInputDialog("Please input the desired
imaginary number for Mu [-1.5, 1.5]");

            promptMuImaginary = Double.parseDouble(prompt);

        }

        catch(NumberFormatException e) {
```

```
            JOptionPane.showMessageDialog(this, "Error, please input the desired
imaginary number for Mu [-1.5, 1.5]");

            String prompt = JOptionPane.showInputDialog("Please input the desired
imaginary number for Mu");

            promptMuImaginary = Double.parseDouble(prompt);

        }

        return promptMuImaginary;

    }


    //-----------------------------------------------------------------------------------------------
--------------//




      //---------------------------------- Methods that implement the zooming
functions----------------------------------------//



    /**
```

Look out a window directly at some object, say the eye of a bird sitting on the branch of a tree. Now draw a rectangle, R1, on the glass with dimensions HxW, centered on the bird's eye. Inside that rectangle, draw a smaller rectangle R2 with dimensions 0.95H x 0.95W, which again is centered on the birds eye. If we scale (i.e., zoom) R2, to produce rectangle R3 such that it has the same size as R1, all the points in R3—except for one—will be shifted relative to R2. The one point that will remain fixed is the one corresponding to the bird's eye: it will remain at the center, because, in this case, it is the zoom's “fixed point.”

For this project, instead of looking out a physical window, we're looking at a region on the complex plane. If R1 is the region depicted by the current frame, R2 would be the region depicted by the next frame.

In this project, the zoom's fixed point will be determined by the position of the mouse pointer. You may choose between three options:

perform an animated translation of the fixed point to the center, and then begin zooming about the image's center [this is what I demoed in class]. (See Javadocs for how to programmatically move the mouse pointer.)

set the fixed point to the mouse pointer's location at the time the mouse button was depressed and zoom [this will produce an offset (eccentric) zoom effect],

let the fixed point follow the mouse pointer during the zoom [effectively giving the user control of a fly-by camera].

```
**/

private void centerImage(double x, double y)
{
    double cx = 300.0, cy = 225.0;

    double xdist = Math.abs(300.0 - x)/300.0;

    double ydist = Math.abs(225.0 - y)/225.0;

    double realDist = Math.abs(secondReal - secondRealFinal)/2.0;

    double imgDist = Math.abs(secondImage - secondImageFinal)/2.0;

    if(x > cx)
    {
```

```
    secondReal += realDist * xdist;

    secondRealFinal += realDist * xdist;
}
else if(x < cx)
{
    secondReal -= realDist * xdist;

    secondRealFinal -= realDist * xdist;
}

if(y > cy)
{
    secondImage -= imgDist * ydist;

    secondImageFinal -= imgDist * ydist;
}
else if(y < cy)
{
    secondImage += imgDist * ydist;

    secondImageFinal += imgDist * ydist;
}

if(isJulia)
```

```
            juliasetImageMaker(secondReal, secondImage, secondRealFinal,
secondImageFinal);


        else


            mandelbrotImageMaker(secondReal, secondImage, secondRealFinal,
secondImageFinal);
    }


    private void zoomIn()
    {
        double diff_real = Math.abs(secondReal - secondRealFinal);


        double diff_img = Math.abs(secondImage - secondImageFinal);


        double tempReal = secondReal;


        double tempImg = secondImage;


        secondReal += diff_real * 0.025;


        secondImage -= diff_img * 0.025;


        secondRealFinal = tempReal + (diff_real * 0.975);
```

```
        secondImageFinal = tempImg - (diff_img * 0.975);

        if(isJulia)

            juliasetImageMaker(secondReal, secondImage, secondRealFinal,
secondImageFinal);

        else

            mandelbrotImageMaker(secondReal, secondImage, secondRealFinal,
secondImageFinal);
    }
    private void zoomOut()
    {
        double diff_real = Math.abs(secondReal - secondRealFinal);

        double diff_img = Math.abs(secondImage - secondImageFinal);

        secondReal -= diff_real * 0.025;

        secondImage += diff_img * 0.025;

        secondRealFinal += diff_real * 0.025;

        secondImageFinal -= diff_img * 0.025;
```

```
        if(isJulia)

            juliasetImageMaker(secondReal, secondImage, secondRealFinal,
secondImageFinal);

        else

            mandelbrotImageMaker(secondReal, secondImage, secondRealFinal,
secondImageFinal);

    }

    //-----------------------------------------------------------------------------------------------
--------------------------------//

    /**
     When the Mandelbrot options is selected, the program shall create and display a
600 x 450 image corresponding the region (-2 + 1.5i) to (2 - 1.5i) after 100 iterations
using one of the coloring schemes described below (you may chose to support both
coloring schemes and let the user chose which to use) [note: the y-axis shall be the
imaginary axis, while the x-axis shall be the real-axis]

     **/
    private void mandelbrotImageMaker(double initA, double initB, double endA,
double endB)
    {
        isJulia = false;
```

```
double horizDistance = Math.abs(endA - initA)/601.0;

double vertDistance = Math.abs(endB - initB)/451.0;

double w = WIDTH/(Math.abs(initA - endA));

double h = HEIGHT/(Math.abs(initB - endB));

Color[] colors = interpolateColors();

//Has a fractal display panel 600 pixels wide and 450 pixels tall

//image = new BufferedImage(600, 450, BufferedImage.TYPE_INT_RGB);

//g2 = (Graphics2D)image.createGraphics();

g2.setColor(Color.WHITE);

g2.fillRect(0, 0, 600, 450);

int tMax = 100, t = 0;

double z_real = 0.0, z_i = 0.0;
```

```
for(int i = 0; i <= 450; i++)

{
    if(i != 0){initB -= vertDistance;
    }

    for(int j = 0; j <= 600; j++)
    {

        if(j != 0){initA += horizDistance;

        }

        z_real = 0.0;

        z_i = 0.0;

        t = 0;

        while(t < tMax)
        {
            double tempZ = z_real;

            z_real = (z_real * z_real) - (z_i * z_i) + initA;
```

```
    z_i = (2.0 * tempZ * z_i) + initB;

    if(((z_real * z_real) + (z_i * z_i)) >= 4)

        break;

    else ++t;
}

/**

 Style 2: dragon tiger
 if ( t == tMax )
 plot pixel with black
 else if t is even
 plot pixel with black
 else
 plot pixel with color[t]

 **/

if(t == tMax)

    g2.setColor(Color.BLACK);
else if(t % 2 == 0)
```

```
                g2.setColor(Color.BLACK);
            else

                g2.setColor(colors[t]);

            g2.draw(new Line2D.Double((initA - secondReal) * w, (initB - secondImage)
* (-h), (initA - secondReal) * w, (initB - secondImage) * (-h)));
          }
          initA = secondReal;
       }
       //displayBufferedImage(image);
       repaint();
   }


   /**
    When the Julia options is selected, the program shall prompt the user for the
value of μ, create and display a 600 x 450 image corresponding to the region (-2 +
1.5i) to (2 - 1.5i) after 100 iterations using one of the coloring schemes described
below. [note: the y-axis shall be the imaginary axis, while the x-axis shall be the real-
axis]

    **/

   private void juliasetImageMaker(double initA, double initB, double endA, double
endB)
```

```
    {
        isJulia = true;

        String mu = "";

        if(!isZooming)
        {
            mu = JOptionPane.showInputDialog("Please input your desired value of mu
[Example: 0.285 + 0.01i]");

            String[] strs = mu.split("- |\\+");

            mu_real = Double.parseDouble(strs[0]);

            mu_i = Double.parseDouble(strs[1].substring(0, strs[1].length() - 1));
        }

        double horizDistance = Math.abs(endA - initA)/601.0;

        double vertDistance = Math.abs(endB - initB)/451.0;

        double w = WIDTH/(Math.abs(initA - endA));

        double h = HEIGHT/(Math.abs(initB - endB));
```

```
Color[] colors = interpolateColors();

//Has a fractal display panel 600 pixels wide and 450 pixels tall

//image = new BufferedImage(600, 450, BufferedImage.TYPE_INT_RGB);

//g2 = (Graphics2D)image.createGraphics();

g2.setColor(Color.WHITE);

g2.fillRect(0, 0, 600, 450);

int tMax = 100, t = 0;

double z_real = initA, z_i = initB;

for(int i = 0; i <= 450; i++)
{
    if(i != 0){z_i -= vertDistance;


    }
    for(int j = 0; j <= 600; j++)
    {
        if(j != 0){z_real += horizDistance;}
```

```
t = 0;

double tempZr = z_real;

double tempZi = z_i;

while(t < tMax)
{
   double tempZ = tempZr;

   tempZr = (tempZr * tempZr) - (tempZi * tempZi) + mu_real;

   tempZi = (2.0 * tempZ * tempZi) + mu_i;

   if(Math.sqrt((tempZr * tempZr) + (tempZi * tempZi)) >= 2)

      break;

   else ++t;
}

/**

 Style 2: dragon tiger
 if ( t == tMax )
```

```
            plot pixel with black
            else if t is even
            plot pixel with black
            else
            plot pixel with color[t]

            **/

            if(t == tMax)

               g2.setColor(Color.BLACK);

            else if(t % 2 == 0)

               g2.setColor(Color.BLACK);
            else

               g2.setColor(colors[t]);

            g2.draw(new Line2D.Double((z_real - secondReal) * w, (z_i - secondImage) *
(-h), (z_real - secondReal) * w, (z_i - secondImage) * (-h)));
         }

         z_real = initA;
      }
```

```
    //displayBufferedImage(image);
    repaint();
  }

  /**
```

Color scheme

The first thing you'll need to do is programmatically populate an array with 100 smoothly interpolated colors. At a minimum there must be at least three control colors (you may have more): starting [color 0], midpoint [color n], and final [color 100]— where n is a constant of your choice such that $20 \le n \le 80$. When plotting pixels, you may use either of these styles:

Style 1: normal

```
if ( t == tMax )
plot pixel with black
else
plot pixel with color[t]
```

Style 2: dragon tiger

```
if ( t == tMax )
plot pixel with black
else if t is even
plot pixel with black
else
```

```
     plot pixel with color[t]



    **/

    // Method that implements interpolating and extracting the colors from the
different bitmap channel interpolations

    private Color[] interpolateColors()
    {
        int color_init = 0xffff0000;

        int color_n = 0xff00ff00;

        int color_end = 0xff006400;

        Color[] array = new Color[100];

        array[0] = new Color(color_init);

        array[50] = new Color(color_n);

        array[99] = new Color(color_end);

        // Steps to interpolate through the color channels
```

```
double stepSize = 1/50.0;

// Implements the color channels

double startRed = (double)((color_init >>> 16) & 0xFF);

double startGreen = (double)((color_init >>> 8) & 0xFF);

double startBlue = (double)(color_init & 0xFF);

double midRed = (double)((color_n >>> 16) & 0xFF);

double midGreen = (double)((color_n >>> 8) & 0xFF);

double midBlue = (double)(color_n & 0xFF);

double endRed = (double)((color_end >>> 16) & 0xFF);

double endGreen = (double)((color_end >>> 8) & 0xFF);

double endBlue = (double)(color_end & 0xFF);

double s1r = startRed, s1g = startGreen, s1b = startBlue;
```

```
// Calculates the new interpolations

double dr = (midRed - startRed)*stepSize;

double dg = (midGreen - startGreen)*stepSize;

double db = (midBlue - startBlue)*stepSize;

for(int i = 1; i < 50; i++)
{
   s1r += dr;

   s1g += dg;

   s1b += db;

   array[i] = new Color((int)s1r, (int)s1g, (int)s1b);
}

s1r = midRed;

s1g = midGreen;

s1b = midBlue;
```

```
        dr = (endRed - midRed)*stepSize;

        dg = (endGreen - midGreen)*stepSize;

        db = (endBlue - midBlue)*stepSize;

        // Returns the new colors

        for(int i = 51; i < 99; i++)
        {
            s1r += dr;

            s1g += dg;

            s1b += db;

            array[i] = new Color((int)s1r, (int)s1g, (int)s1b);
        }

        return array;
    }

    // When the Save image options is selected, the program shall prompt the user for the output file and save the current

    // Mandelbrot or julia set image as a PNG file and a desired image name
```

```
    private void saveTheImage()
    {
        // Prompts the user to enter a desired file name

        String savingFileName = (String)JOptionPane.showInputDialog("Enter the desired
name for the PNG file you'd like to save");

        // Saves the file as a png (portable network graphics)
        savingFileName += ".png";

        File outputFile = new File(savingFileName);
        try
        {
            javax.imageio.ImageIO.write( image, "png", outputFile );
        }
        catch ( IOException e )
        {
            JOptionPane.showMessageDialog( ImageFrame.this,
                                           "Error saving file",
                                           "oops!",
                                           JOptionPane.ERROR_MESSAGE );
        }
    }
```

```
    // Displays our end Buffered Image

    public void displayBufferedImage(BufferedImage image)
    {

        this.setContentPane(new JLabel(new ImageIcon(image)));

        this.validate();
    }

    public void paintComponent(Graphics g)
    {
        this.paintComponent( g);
        g.drawImage (image, 0,0, null);
    }
}
```