Sahar K Hussain
CAP3027
Section 1925
October 23rd, 2015
HW09 + Bonus

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

Sahar K Hussain

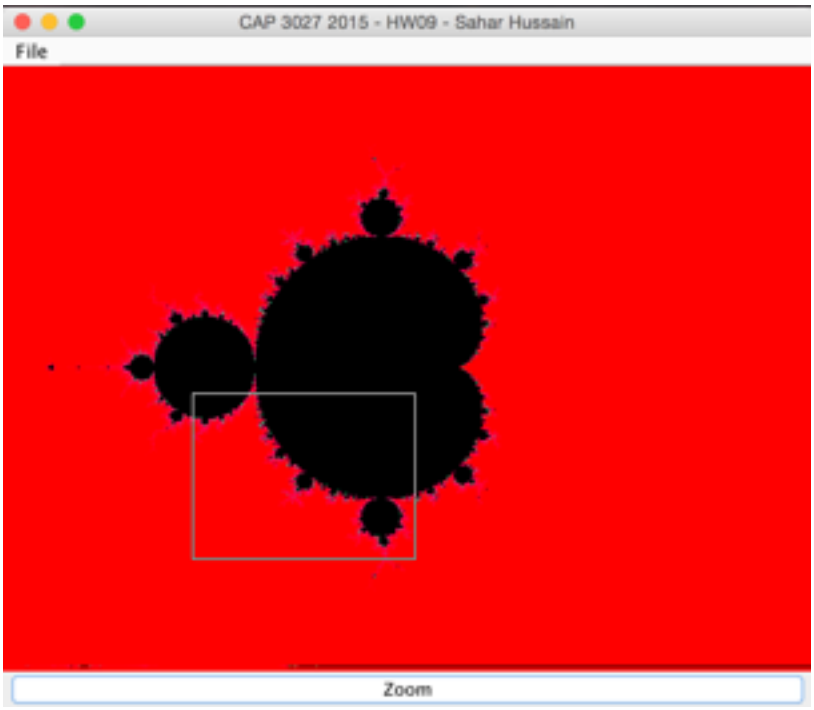
The easiest part of the homework was building a little bit off of my last homework and interpreting how to compute the fractals. The hardest part was actually figuring out how to make the mandelbrot image for me. I believe the assignment's educational objective was to help us practice using new techniques of java. It was gratifying after figuring out how to compute the image.

Sahar K Hussain
CAP3027
Section 1925
October 23rd, 2015
HW09 + Bonus

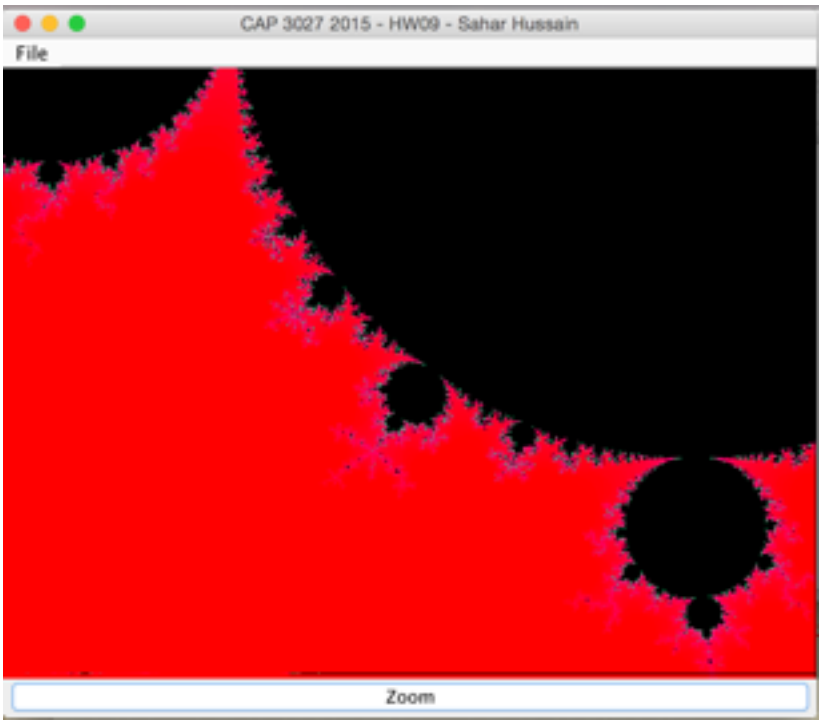
Standard Program Deliverables:
1) Yes, the program compiles without errors.
2) Yes, the program compiles without warnings.
3) Yes, the program runs without crashing.
4) I tested the program on my personal laptop and utilized xCode for inputting the code and the terminal for compiling and running the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

Screenshots:

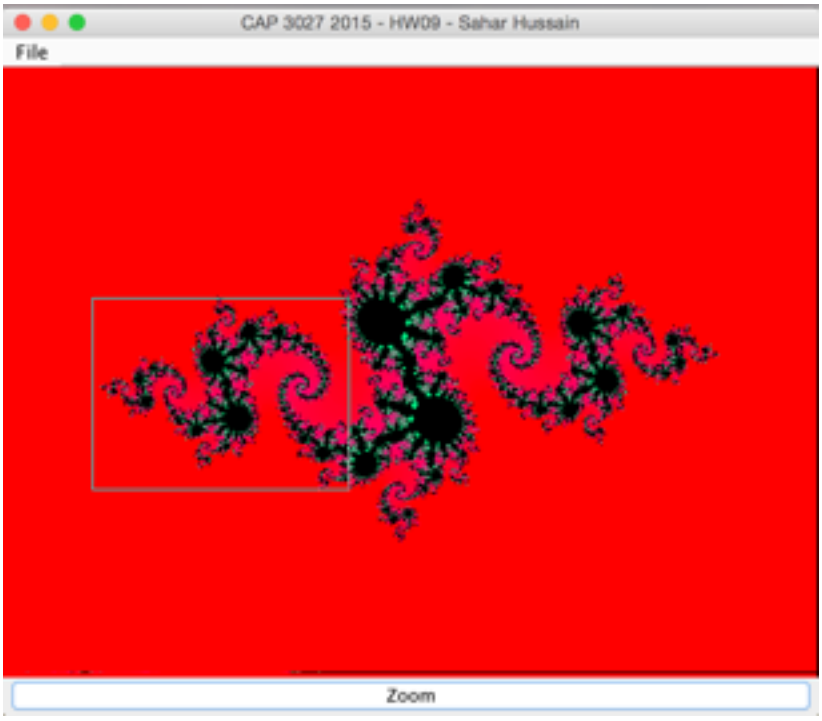
Mandelbrot Unzoomed:



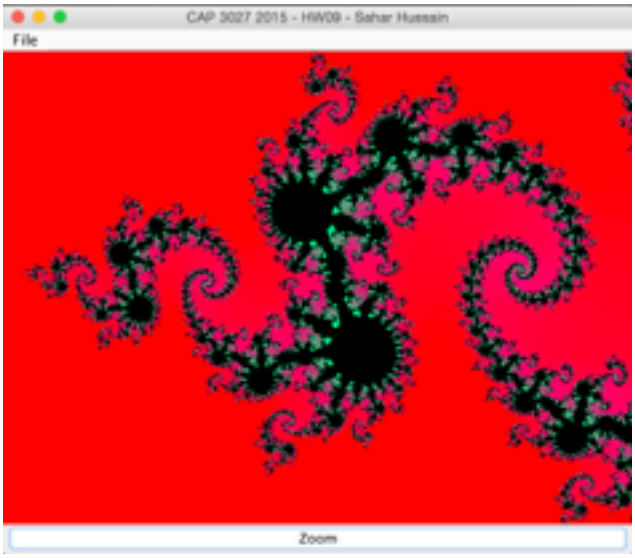
MandelbrotZoomed:



JuliaUnzoomed:



JuliaZoomed:



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

//HW09 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

 **/


// 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.*;
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;



public class DisplayImage
{
   // Has an area select 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();
         }
      });
   }

   public 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
{
    // Creates the new areaselectpanel calling object from AreaSelectPanel.java
class
    // AreaSelectPanel.java
    // - click and drag to select rectangular area on an image
    private AreaSelectPanel panel;

    // Initializes our variables
    private JButton button;

    private int gridWidth = 600;

    private int gridHeight = 450;

    private double x = 0;
```

```
private double y = 0;

private int tmax = 100;

private boolean mandelbrot;

private BufferedImage image = null;

private int [] finalColorScheme = new int [100];

private double firstR = -2;

private double secondR = 2;

private double changeReal = secondR-firstR;

private double firstImagine = -1.5;

private double secondImagine = 1.5;

private double changeImaginary = secondImagine-firstImagine;

private double[] variable = new double[2];

//  Constructor
```

```
public ImageFrame(int gridWidth, int gridHeight)
{
   //setup the frame's attributes

   this.setTitle( "CAP 3027 2015 - HW09 - Sahar Hussain" );
   this.setSize (gridWidth, gridHeight);

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

   image = newImageCreator(gridWidth,gridHeight);

   panel = new AreaSelectPanel( image);

   //Has a button at the bottom of the frame labeled "Zoom", which when
clicked, zooms into the selected area.

   button = new JButton( "Zoom" );
   button.addActionListener( new ActionListener()
                     {
      public void actionPerformed( ActionEvent event )
      {
         updateImage(mandelbrot);
      }
```

```
        } );

        this.getContentPane().add( panel, BorderLayout.CENTER );

        this.getContentPane().add( button, BorderLayout.SOUTH );

        this.pack();

        this.setVisible( true );

        //setup the file chooser dialog

        //chooser = new JFileChooser();
        //chooser.setCurrentDirectory( new File( "." ) );

    }

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

    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)
   {
      firstR = -2;
      secondR = 2;
      changeReal = secondR-firstR;
      firstImagine = -1.5;
      secondImagine = 1.5;
      changeImaginary = secondImagine-firstImagine;
      mandelbrotImageMaker(firstR,firstImagine,secondR,secondImagine);
   }
} );

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)
        {
            variable = userMuPrompter();
            firstR = -2;
            secondR = 2;
            changeReal = secondR-firstR;
            firstImagine = -1.5;
            secondImagine = 1.5;
            changeImaginary = secondImagine-firstImagine;
            juliasetImageMaker(firstR,firstImagine,secondR,secondImagine,
variable[0], variable[1]);
        }
    } );

    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 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 Mandelbrot Creations
-----------------------------------------//


    /**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 mandelbrotRealFirst, double
mandelbrotImagineFirst, double mandelbrotRealSecond, double
mandelbrotImagineSecond)

   {

       colorInterpolationChange();


       changeReal = mandelbrotRealSecond- mandelbrotRealFirst;


       changeImaginary = mandelbrotImagineSecond - mandelbrotImagineFirst;


       double stepsReal = (mandelbrotRealSecond- mandelbrotRealFirst)/(gridWidth
- 1);


       double stepsImagine = (mandelbrotImagineSecond - mandelbrotImagineFirst)/
(gridHeight -1);


       double real = mandelbrotRealFirst;




          for(int x = 0; x < gridWidth; x++)

          {

             double img =  mandelbrotImagineSecond;
```

```
            for(int y = 0; y < gridHeight; y++)
            {
                double [] m_secondNumber = new double [2];

                m_secondNumber[0] = 0;

                m_secondNumber[1] = 0;

                int t = 0;

                // Divergence Test
                /**You may optionally support, in addition to the standard divergence
test [diverged when z's magnitude is ≥ 2] the one Laurens Lapré discovered, via a
typo: when the relationship ((realSquared - imgSquared) < 4) is false, z is
considered to have diverged. It's an interesting alternative that produces fractals
with Kirlian-like “auras.”
                 **/

                while(t!=tmax)
                {
                    m_secondNumber = muAddition(m_secondNumber[0],
m_secondNumber[1], real, img);

                    if(squareCalc(m_secondNumber[0],m_secondNumber[1]) > 4.0)
                    {
                       break;
                    }
```

```
            else
            {
                ++t;
            }
        }

        if(t == tmax)
        {
            // Plots the pixel with black

            double [] newBitmap = new double [2];

            newBitmap = coordinateBit(real, firstR, secondR, img, firstImagine,
secondImagine);

            image.setRGB((int)(newBitmap[0]),(int)(newBitmap[1]),
0xFF000000);
        }

        else

        {
            double [] newBitmap = new double [2];
```

```
                newBitmap = coordinateBit(real, firstR, secondR, img, firstImagine,
secondImagine);


                image.setRGB((int)(newBitmap[0]),(int)
(newBitmap[1]),finalColorScheme[t]);
            }


            img -= stepsImagine;


        }


        real += stepsReal;
    }


    //updateImage();


    SwingUtilities.invokeLater(new Runnable()
                    {
        public void run()
        {
            mandelbrot = true;


            panel.setImage(image);
        }
    });
}
```

```
   //-------------------------------------------------------------------------------------------------------------
----------------------//


   //---------------------------------- Methods that implement the JuliaSet
Creations-----------------------------------------//


   /**


    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 juliaFirstReal, double juliaImagineFirst,
double juliaSecondReal, double juliaImagineSecond, double juliaRealImager,
double juliaImagineImager)
   {
      double [] m_firstNumber = new double [2];


      m_firstNumber[0] = juliaRealImager;


      m_firstNumber[1] =  juliaImagineImager;


      colorInterpolationChange();
```

```
        changeReal = juliaSecondReal- juliaFirstReal;

        changeImaginary = juliaImagineSecond - juliaImagineFirst;

        double stepsReal = (juliaSecondReal- juliaFirstReal)/(gridWidth - 1);

        double stepsImagine = (juliaImagineSecond - juliaImagineFirst)/(gridHeight
-1);

        double real = juliaFirstReal;

        for(int x = 0; x < gridWidth; x++)
        {

           double img =  juliaImagineSecond;

           for(int y = 0; y < gridHeight; y++)
           {
              double [] m_secondNumber = new double [2];

              m_secondNumber[0] = real;

              m_secondNumber[1] = img;
```

```
int t = 0;


// Divergence Test

/**You may optionally support, in addition to the standard divergence test [diverged when z's magnitude is ≥ 2] the one Laurens Lapré discovered, via a typo: when the relationship ((realSquared - imgSquared) < 4) is false, z is considered to have diverged. It's an interesting alternative that produces fractals with Kirlian-like “auras.”

 **/


   while(t!=tmax)

   {

      m_secondNumber = muAddition(m_secondNumber[0], m_secondNumber[1], m_firstNumber[0], m_firstNumber[1]);


      if(squareCalc(m_secondNumber[0], m_secondNumber[1]) > 4.0)

      {


         break;

      }


      else

      {

         ++t;

      }

   }
```

```
                if(t ==tmax)
                {
                   //Plot pixel with black

                   double [] newBitmap = new double [2];

                   newBitmap =
coordinateBit(real,firstR,secondR,img,firstImagine,secondImagine);

                   image.setRGB((int)(newBitmap[0]),(int)(newBitmap[1]),
0xFF000000);

                }

                else
                   //Plot pixel with color
                {
                   double [] newBitmap = new double [2];

                   newBitmap =
coordinateBit(real,firstR,secondR,img,firstImagine,secondImagine);

                   image.setRGB((int)(newBitmap[0]),(int)
(newBitmap[1]),finalColorScheme[t]);

                }
```

```
            img -= stepsImagine;

        }

        real += stepsReal;

    }
    SwingUtilities.invokeLater(new Runnable()
        {
        public void run()
        {
            mandelbrot = false;
            panel.setImage(image);
        }
    });
  }

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

  private double [] muAddition(double secondReal, double secondImagine, double
firstReal, double firstImagine)
  {
    double [] calculator = new double [2];
```

```
        calculator[0] = (secondReal*secondReal) - (secondImagine*secondImagine)+
firstReal;


        calculator[1] = (2*secondReal*secondImagine) + firstImagine;


    return calculator;
    }


    private double squareCalc(double realSecondImage, double imageSecondImage)
    {
        double calculator = 0.0;


        calculator = realSecondImage*realSecondImage +
imageSecondImage*imageSecondImage;


    return calculator;


    }


    private double [] coordinateBit(double realFirstImage, double changefirstR,
double changeSecondR, double imageFirstImage, double changeFirstImagine,
double changeSecondImagine)
    {
        double calculator [] = new double [2];


        double changeRealCalculate = changeSecondR-changefirstR;
```

```
        double changeImaginaryCalculate = changeSecondImagine-
changeFirstImagine;


        calculator[0] = ((realFirstImage - changefirstR ) / changeRealCalculate*
(gridWidth-1));


        calculator[1] = (imageFirstImage - changeFirstImagine ) /
changeImaginaryCalculate * (gridHeight-1);


    return calculator;

    }


    private void colorInterpolationChange()

    {

        int totalColorChannel = 0;


        double[] leftColor;


        double[] rightColor;


        double changeRedChannel;


        double changeGreenChannel;


        double changeBlueChannel;
```

```
double channelRed; // Implements the Red Color channel

double channelGreen; // Implements the Green Color Channel

double channelBlue; // Implements the Blue Color Channel

leftColor = colorExtract(16711680);

rightColor = colorExtract(16711795); // A beautiful pink decimal value!
#ff0073

changeRedChannel = (rightColor[1] - leftColor[1])/(49);

changeGreenChannel = (rightColor[2] - leftColor[2])/(49);

changeBlueChannel = (rightColor[3] - leftColor[3])/(49);

channelRed = leftColor[1]; // Creates an array that starts the population of
colors from the left side for the red Channel

channelGreen = leftColor[2]; // Creates an array that starts the population of
colors from the left side for the green Channel

channelBlue = leftColor[3]; // Creates an array that starts the population of
colors from the left side for the blue Channel
```

```
for(int x = 0; x < 49;x++)
{
   channelRed = channelRed + changeRedChannel;

   channelGreen = channelGreen + changeGreenChannel;

   channelBlue = channelBlue + changeBlueChannel;

   if(channelRed > 255)
   {
      channelRed = 255;
   }

   if(channelRed < 0)
   {
      channelRed = 0;
   }

   if(channelGreen > 255)
   {
      channelGreen = 255;
   }

   if(channelGreen < 0)
```

```
        {
            channelGreen = 0;
        }

        if(channelBlue > 255)
        {
            channelBlue = 255;
        }

        if(channelBlue < 0)
        {
            channelBlue = 0;
        }

            totalColorChannel =
changetoIntColors(255,channelRed,channelGreen,channelBlue);

            finalColorScheme[x] =totalColorChannel;//record the color information
in the colorArray for future use                                    }
        }

        leftColor = colorExtract(16711795); // A beautiful pink decimal value!
#ff0073

        rightColor = colorExtract(65420); // A beautiful mint decimal value!
#00ff8c
```

```
changeRedChannel = (rightColor[1] - leftColor[1])/(49);

changeGreenChannel = (rightColor[2] - leftColor[2])/(49);

changeBlueChannel = (rightColor[3] - leftColor[3])/(49);

channelRed = leftColor[1]; // Creates an array that starts the population of
colors from the left side for the red Channel

channelGreen = leftColor[2]; // Creates an array that starts the population
of colors from the left side for the green Channel

channelBlue = leftColor[3]; // Creates an array that starts the population
of colors from the left side for the blue Channel

   for(int x = 49; x < 100 ; x++)
   {

      channelRed = channelRed + changeRedChannel;

      channelGreen = channelGreen + changeGreenChannel;

      channelBlue = channelBlue + changeBlueChannel;

         if(channelRed > 255)
```

```
{
   channelRed = 255;
}

if(channelRed < 0)
{
   channelRed = 0;
}

if(channelGreen > 255)
{
   channelGreen = 255;
}

if(channelGreen < 0)
{
   channelGreen = 0;
}

if(channelBlue > 255)
{
   channelBlue = 255;
}

if(channelBlue < 0)
```

```
                {
                    channelBlue = 0;
                }

            totalColorChannel =
changetoIntColors(255,channelRed,channelGreen,channelBlue);

            finalColorScheme[x] =totalColorChannel;
        }

    }

    // Method that implements extracts the colors from the different bitmap
channel interpolations
    private static double[] colorExtract(int ARGB_)
    {

        double[] colorExtractionArray;

        colorExtractionArray = new double[4];

        colorExtractionArray[0] = ARGB_>>>24;

        colorExtractionArray[1] = (ARGB_<<8) >>> 24;

        colorExtractionArray[2] = (ARGB_<<16)>>>24;
```

```
        colorExtractionArray[3] = (ARGB_<<24)>>>24;


        return (colorExtractionArray);
    }

    // Converts the extracted colors hexidecimal into the int colors


    private int changetoIntColors(double alphaChannel, double redChannel, double
greenChannel, double blueChannel)
    {


        return ((((int)alphaChannel)<<24)|(((int)redChannel)<<16)|((int)
(greenChannel)<<8)|((int)blueChannel));
    }


    private double [] userMuPrompter()
    {


        double [] varTemp = new double[2];


        double [] wrongInput = new double[2];


        wrongInput[0] = -100000;


        wrongInput[1] = 100000;
```

```
    // Calls the methods that prompt the user for the real and imaginary inputs
for Mu

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

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

    // Checks to see if the methods that were prompted are valid inputs

    if(inputChecker(promptMuReal) && inputChecker(promptMuImaginary))
    {

       // Stores them in a temporary variable

       varTemp[0] = Double.parseDouble(promptMuReal);

       varTemp[1] = Double.parseDouble(promptMuImaginary);

    return varTemp;

    }
```

```
        // If the prompts are empty, the program returns an error and exits

    else if (promptMuReal == null || promptMuImaginary == null)
    {

        System.exit(0);

        return wrongInput;

    }

    else
    {
    // If not, we will prompt the user for the Mu

    return userMuPrompter();
    }
}

private boolean inputChecker(String input_)
    {

        try
        {
            double num = Double.parseDouble(input_);
```

```
            if(num<-1000 || num > 1000)
            {
               JOptionPane.showMessageDialog(null, "Oop! Invalid Input", "alert",
JOptionPane.ERROR_MESSAGE);
               return false;
            }
            return true;
            }

         catch(NumberFormatException e)
         {

         JOptionPane.showMessageDialog(null, "Oops! Invalid Input", "alert",
JOptionPane.ERROR_MESSAGE);

         return false;
         }
      }

   protected BufferedImage newImageCreator(int newWidth,int newHeight)
   {

      while (true)
      {
         if (newWidth < 0 || newHeight < 0)
```

```
            return null;
        try
        {
            BufferedImage img = new
BufferedImage(newWidth,newHeight,BufferedImage.TYPE_INT_RGB);

            return img;

        }
        catch (OutOfMemoryError err)
        {
              JOptionPane.showMessageDialog(this, "Oops! Memory error. Utilize
smaller image dimensions");
        }
      }
   }

   public void updateImage(boolean mandelbrot_){

        double theNewR = panel.getUpperLeft().getX() * changeReal + firstR;

        double theNewRsecond = panel.getLowerRight().getX()* changeReal +
firstR;

        double theNewImagine = panel.getUpperLeft().getY() * changeImaginary +
firstImagine;
```

```
        double theNewImagineSecond = panel.getLowerRight().getY() *
changeImaginary + firstImagine;


        firstR = theNewR;


        secondR = theNewRsecond;


        firstImagine = theNewImagine;


        secondImagine = theNewImagineSecond;


        if(mandelbrot_)
        {
           // Calls the method for the mandelbrot
           mandelbrotImageMaker(firstR,firstImagine,secondR,secondImagine);


        }
        else
        {
           // Calls the method for the Julia
           juliasetImageMaker(firstR, firstImagine, secondR,
secondImagine,variable[0], variable[1]);
        }
     }
```

```
   // 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


   public 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 );

      }
```

```
    }


}
```