Sahar K Hussain
CAP3027
Section 1925
October 12th, 2015
HW07

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

Sahar K Hussain

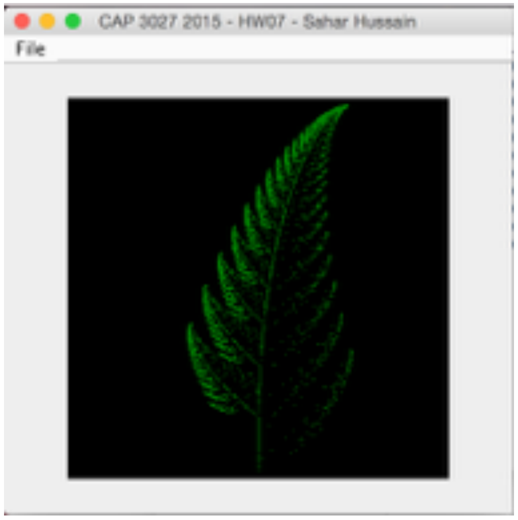
The easiest part of the homework was to bring back old concepts of using a scanner and reading in the lines of the input file and setting up the whole application. The hardest point was to do the transforming because I wasn't entirely sure how to go about flipping my image. I believe the assignment's educational objective was to help us practice using affine transform matrices in an applied java sense and see how our fractals can come to life using them!

Sahar K Hussain
CAP3027
Section 1925
October 12th, 2015
HW07

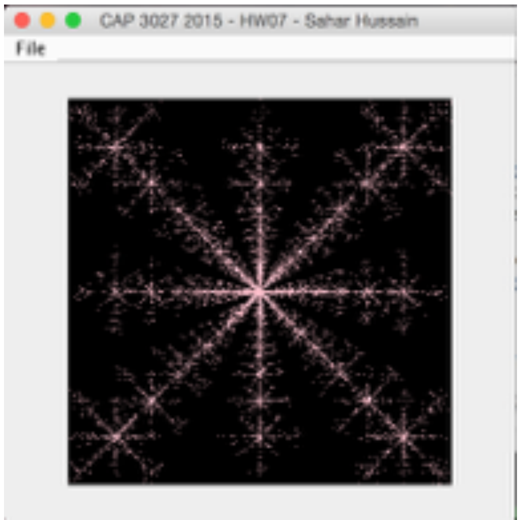
Standard Program Deliverables:
1) Yes, the program compiles without errors.
2) Yes, the program compiles without warnings.
3) Yes, the program runs without crashing.
4) I tested the program on my personal laptop and utilized xCode for inputting the code and the terminal for compiling and running the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

Screenshots:

Barnsley's Fern (Black background - Green Foreground):



Snow-flake Crystal from Pg 107 of the Computational Beauty of Nature book (input file provided):



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

//HW07 Modification to original HW00 code and work by Sahar KH


/**For this assignment, I shall be implementing one of the Affine Transform Fractal
Techniques

as described in Lecture: IFS.

**/


// 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.Point2D;
import java.awt.*;

import java.awt.event.*;

import java.awt.image.*;

import java.io.*;

import javax.imageio.*;

import javax.swing.*;

import java.util.*;

import java.util.ArrayList;

import java.io.IOException;

import java.io.FileReader;

import java.io.BufferedReader;



public class DisplayImage

{

    private static final int WIDTH = 400;

    private static final int HEIGHT = 400;


    // 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
{
    private final JFileChooser chooser;
    private BufferedImage image = null;
    private static int lineCounter = 0;
    private static ArrayList<String> linesInFile = new ArrayList<String>();
    private static ArrayList<AffineTransform> transform = new
ArrayList<AffineTransform>();
    private static ArrayList<Double> seventhProbability = new ArrayList<Double>();
    private static ArrayList<Double> determinantArray = new ArrayList<Double>();
    private static boolean probability = false;
    private static int newBackgroundColor;
    private static int newForegroundColor;
    private static String backgroundColor = "0xFFFFFFFF";
    private static String foregroundColor = "0xFFFFFFFF";
    private static Graphics2D g2d = null;
    private static BufferedImage theNewImage = null;
    private static double sumOfDeterminants = 0;
    private static Point2D.Double firstPoint;
    private static Point2D.Double secondPoint;
    private static double x = 0;
    private static double y = 0;
    private static int width;
    private static int height;

    //constructor

    public ImageFrame(int width, int height)
    {
        //setup the frame's attributes

        this.setTitle( "CAP 3027 2015 - HW07 - Sahar Hussain" );
        this.setSize (width, height);

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

```
        //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 the IFS description


        JMenuItem loadOurIFS = new JMenuItem( "Load IFS description" );

        loadOurIFS.addActionListener( new ActionListener()

                                {

            public void actionPerformed(ActionEvent event)

            {

                loadTheImage();
    }
} );

fileMenu.add( loadOurIFS );

// The JMenuItem that will configure the image

JMenuItem configureOurIFS = new JMenuItem( "Configure image" );
configureOurIFS.addActionListener( new ActionListener()
                          {
    public void actionPerformed(ActionEvent event)
    {
        configureTheImage();
    }
} );

fileMenu.add( configureOurIFS );

// The JMenuItem that will display the image

JMenuItem dispOurImage = new JMenuItem( "Display IFS" );
dispOurImage.addActionListener( new ActionListener()
                          {
    public void actionPerformed(ActionEvent event)
    {
        displayTheImage();
    }
} );

fileMenu.add( dispOurImage );

// 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);

   }


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


   // Method that opens the image selected by the user

   // Prompts the user for the file containing the IFS's description (JFileChooser)

   // Loads IFS description from configuration file


   private void loadTheImage()

   {
```

```
File info = null;


if (chooser.showOpenDialog(this) == JFileChooser.APPROVE_OPTION)
{
    info = chooser.getSelectedFile();
}


String line = "";


try
{
    // Takes in the IFS file and reads the lines


    BufferedReader reader = new BufferedReader(new FileReader(info));


    while( (line = reader.readLine() ) != null)
    {
        lineCounter++; // the number of lines
        linesInFile.add(line); // adds the lines read to our array list


    }


    MatrixCalculator(); // Goes to calculating the affine matrix
    reader.close();
}
```

```
        // If there is nothing in the file, Display our error message
        catch (IOException exception)
        {
            JOptionPane.showMessageDialog(ImageFrame.this, "Error loading IFS
description file", "oops!", JOptionPane.ERROR_MESSAGE );
        }
    }


    public void MatrixCalculator()
    {
        for (int m = 0; m < lineCounter; m++) // goes through the number of lines
        {
            Scanner scanner = new Scanner(linesInFile.get(m)); // gets the lines from the
File from our array list


            // Initalizes values for the 6 Inputs given
            double m00 = scanner.nextDouble();
            //System.out.println(m00);
            double m01 = scanner.nextDouble();
            //System.out.println(m01);
            double m10 = scanner.nextDouble();
            //System.out.println(m10);
            double m11 = scanner.nextDouble();
            //System.out.println(m11);
            double m02 = scanner.nextDouble();
            //System.out.println(m02);
            double m12 = scanner.nextDouble();
            //System.out.println(m12);

            // Representing the transform matrix
            AffineTransform calculateAffine = new
AffineTransform(m00,m10,m01,m11,m02,m12);
            transform.add(calculateAffine);

            // If we find lines in the file that consists of 7 numbers
            // Initialize a value for the 7th input too (the probability)

            if(scanner.hasNextDouble() == true)
            {
                seventhProbability.add(scanner.nextDouble());
                probability = true;
            }
        }
    }

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

    // Prompts the user for the desired image's width
    private int userWidth() {
        int promptWidth = 0;
        try {
            String prompt = JOptionPane.showInputDialog("Please input the desired image
width you would like");
            promptWidth = Integer.parseInt(prompt);
        }
        catch(NumberFormatException e) {
            JOptionPane.showMessageDialog(this, "Error, please input the desired image
width you would like");
            String prompt = JOptionPane.showInputDialog("Please enter the desired image
width");
            promptWidth = Integer.parseInt(prompt);
        }
        return promptWidth;
    }


    // Prompts the user for the desired image's height
    private int userHeight() {
        int promptHeight = 0;
        try {
            String prompt = JOptionPane.showInputDialog("Please input the desired image
height you would like");
            promptHeight = Integer.parseInt(prompt);
        }
        catch(NumberFormatException e) {
            JOptionPane.showMessageDialog(this, "Error, please input the desired image
height you would like");
```

```
            String prompt = JOptionPane.showInputDialog("Please enter the desired image
height");

            promptHeight = Integer.parseInt(prompt);

        }

        return promptHeight;

    }



    // Prompts the user for the desired number of generations

    private int userGenerations() {

        int promptGenerations = 0;

        try {

            String prompt = JOptionPane.showInputDialog("Please input the desired
number of generations you would like");

            promptGenerations = Integer.parseInt(prompt);

        }

        catch(NumberFormatException e) {

            JOptionPane.showMessageDialog(this, "Error, please input the desired number
of generations you would like ");

            String prompt = JOptionPane.showInputDialog("Please enter the desired
number of generations");

            promptGenerations = Integer.parseInt(prompt);

        }

        return promptGenerations;

    }



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

```
    // Calls the methods that prompt the user for the desired image's width and height

    // Prompts the user for the desired image's background & foreground colors as
hexadecimal colors

    // Creates a buffered image


    public void configureTheImage()

    {

        // Calls the methods that prompt for the user's desired width and height

        this.width = userWidth();

        this.height = userHeight();


        // Allows the user to enter a desired background and foreground colors

        backgroundColor = (String)JOptionPane.showInputDialog("Enter the desired
background color; Ex. FF2200");

        foregroundColor = (String)JOptionPane.showInputDialog("Enter the desired
foreground color; Ex. FFFFFF");


        // Converts the Colors & parsing

        newBackgroundColor = (int)Long.parseLong( backgroundColor.substring( 0,
backgroundColor.length() ), 16 );

        newForegroundColor = (int)Long.parseLong( foregroundColor.substring( 0,
foregroundColor.length() ), 16 );

        newBackgroundColor = newBackgroundColor | 0xFF000000;

        newForegroundColor = newForegroundColor | 0xFF000000;


        // Creates a new Buffered Image
        image = new BufferedImage(width, height, image.TYPE_INT_ARGB);
        g2d = (Graphics2D) image.createGraphics();

        // Sets the Background and the foreground colors
        Color color = new Color(newBackgroundColor);
        g2d.setColor(color);
        g2d.fill(new Rectangle(0,0,width,height));
        Color secondColor = new Color(newForegroundColor);
        g2d.setColor(secondColor);

    }

    // Displays the Image after Configuration
    // Prompts the user for the number of generations, n
    // Generates the n-th generation image

    public void displayTheImage()
    {
        // Calls the method that prompt for the user's desired number of generations
        int numOfGenerations = userGenerations();

        // Calls the methods that prompt for the user's desired width and height
        //this.width = userWidth();
        //this.height = userHeight();
```

```
        // Gets the inital x and y positions

        x = 0.555;

        y = 0.555;


        firstPoint = new Point2D.Double(x,y);

        secondPoint = new Point2D.Double(x,y);


        // If we weren't given the probability (the 7th input) , we calculate the
probability using the determinant

        // The absolute value of each matrix's determinant and then dividing each by
the sum of all the determinants


        if(probability == false)

        {

            // Calls the determinant calculator method

            getDeterminant();

            for( int i = 0; i < lineCounter; i++)

            {

                seventhProbability.add(determinantArray.get(i)/sumOfDeterminants);

            }

        }


        // For the user's desired number of generations, implement the image
transformation


        for ( int i = 0; i < numOfGenerations; i++)
        {
            transformTheImage();

            // Makes sure y is in the correct direction by flipping the image

            theNewImage = new BufferedImage(width, height, 
theNewImage.TYPE_INT_ARGB);
            for (int k = 0; k < width; k++)
            {
                for (int j = 0; j < height; j++)
                {
                    theNewImage.setRGB( k, j, image.getRGB ( k , height - j - 1));

                }
            }
        }

        displayBufferedImage(theNewImage);

    }

    // Determinant Calculator
    public void getDeterminant()
    {
        for (int i = 0; i < lineCounter; i++)
        {
            if((transform.get(i).getDeterminant()) > 0)
            {
                determinantArray.add(transform.get(i).getDeterminant());
                sumOfDeterminants += transform.get(i).getDeterminant();
            }
            else
            {
                // If the values of the determinant are zero
                determinantArray.add(0.01);
                sumOfDeterminants += .01;
            }
        }
    }

    // Transforms the Image
    public void transformTheImage()
    {
        double random = Math.random();
        double previous = 0;
        for (int i = 0; i < lineCounter; i++)
        {
            double store = seventhProbability.get(i);
            // checks if random value is in range of the probability
            if (random <= store + previous)
            {
                AffineTransform newMatrix = transform.get(i);


                newMatrix.transform(firstPoint, secondPoint);


                if((secondPoint.getX()) <=1 && secondPoint.getY() <= 1 &&
    (secondPoint.getY()) >= 0 && (secondPoint.getX()) >= 0)
                {
                    // //set pixel if in bounds
                    image.setRGB((int)(secondPoint.getX()*this.width),(int)
    (secondPoint.getY()*this.height),newForegroundColor);
                }
                firstPoint = secondPoint;
                break;
            }


            previous += store;
        }
    }


    //displays our end BufferedImage


    public void displayBufferedImage( BufferedImage image )
    {


        //this.setContentPane( new JScrollPane(new JLabel(new ImageIcon (image) ) ) );
```

```
        //this.validate();
        ImageIcon icon = new ImageIcon();
        JLabel label = new JLabel(icon);
        icon.setImage( image );
        label.repaint();
        this.setContentPane(label);
        validate();


    }


    // Saves the Image


    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( theNewImage, "png", outputFile );
        }
        catch ( IOException e )
        {
            JOptionPane.showMessageDialog( ImageFrame.this,
                                           "Error saving file",
                                           "oops!",
                                           JOptionPane.ERROR_MESSAGE );
        }
    }


}
```