Sahar K Hussain
CAP3027
Section 1925
October 19th, 2015
HW08

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 the inputs and outputs of files as a result of getting a lot more practice with that. The hardest part was actually figuring out how to generate a string to calculate reading in the files and using the dictionary java docs, it was a fairly new concept to me. I believe the assignment's educational objective was to help us practice using new techniques of java and applying it to reading/outputting strings so we can learn to read in text files and produce a cool image!

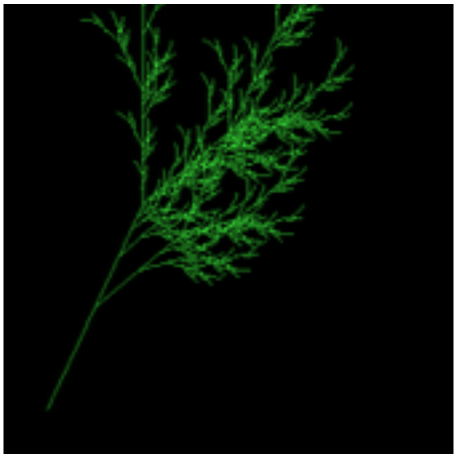
Sahar K Hussain
CAP3027
Section 1925
October 19th, 2015
HW08

Standard Program Deliverables:

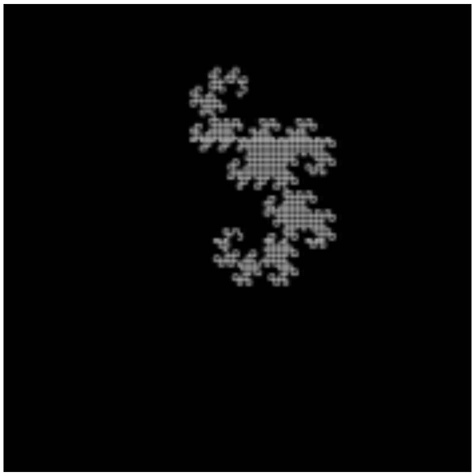
1) Yes, the program compiles without errors.
2) Yes, the program compiles without warnings.
3) Yes, the program runs without crashing.
4) I tested the program on my personal laptop and utilized xCode for inputting the code and the terminal for compiling and running the program.
5) The program does meet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

Screenshots:

Fractal Plant (Black background - Green Foreground):



Dragon Curve (Black background - White Foreground):



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
//HW08 Modification to original HW00 code and work by Sahar KH

/**For this assignment, I shall be implementing one of the L-System/Turtle Graphics Techniques
 as described in Lecture: L-Systems
 **/

// 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 g2;
    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 double inputDelta;
    private int inputScaling;
    private String inputInitiator;
    private static int width;
    private static int height;
    private String[] LSinstruct;

    //constructor


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


        this.setTitle( "CAP 3027 2015 - HW08 - 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 L-System description


        JMenuItem loadOurLS = new JMenuItem( "Load L-System" );

        loadOurLS.addActionListener( new ActionListener()
                             {
    public void actionPerformed(ActionEvent event)
    {
        loadTheImage();
    }
} );

fileMenu.add( loadOurLS );

// The JMenuItem that will configure the image

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

fileMenu.add( configureOurLS );

// The JMenuItem that will display the image

JMenuItem dispOurLS = new JMenuItem( "Display L-System" );
```

```
dispOurLS.addActionListener( new ActionListener()
                              {
    public void actionPerformed(ActionEvent event)
    {
        displayTheImage();
    }
} );

fileMenu.add( dispOurLS);

// 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 L-System's Description
(JFileChooser)
```

```
// Loads L-system description from configuration file

private void loadTheImage()
{
    File info = null;

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

    Scanner scanner = null;
    try
    {
        scanner = new Scanner(info);
    }
    catch(FileNotFoundException exception){}

    int lineCounter = 0;

    while(scanner.hasNextLine())
```

```
{
    scanner.nextLine();
    lineCounter++;
}
LSinstruct = new String[lineCounter - 3];

    // Takes in the LS file and reads the lines

BufferedReader reader = null;

try{
    reader = new BufferedReader(new FileReader(info));
}
catch(FileNotFoundException exception){}
try{

    // Takes in the delta - in degrees
    inputDelta = Double.parseDouble(reader.readLine());

    // Takes in the segment length scaling factor
    inputScaling = Integer.parseInt(reader.readLine());

    // Takes in the initiator string
    inputInitiator = reader.readLine();
```

```
            // Takes in up to 10 production rules (one per line)
            for (int m = 0; m < LSinstruct.length; m++)
            {
                LSinstruct[m] = reader.readLine();
            }
        }

        // 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 );
        }
    }

    //--------------------------------- 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;
    }


    // Prompts the user for the turtle's initial state - X Position
    private double userXPosition() {
        double promptXPosition = 0;
        try {
            String prompt = JOptionPane.showInputDialog("Please input the desired initial X Position [-1.0, 1.0]");
            promptXPosition = Double.parseDouble(prompt);
        }
        catch(NumberFormatException e) {
            JOptionPane.showMessageDialog(this, "Error, please input the desired initial X Position [-1.0, 1.0]");
            String prompt = JOptionPane.showInputDialog("Please enter the desired initial X Position [-1.0, 1.0]");
            promptXPosition = Double.parseDouble(prompt);
        }
        return promptXPosition;
    }


    //  Prompts the user for the turtle's initial state - Y Position
    private double userYPosition() {
        double promptYPosition = 0;
        try {
            String prompt = JOptionPane.showInputDialog("Please input the desired initial Y Position [-1.0, 1.0]");
            promptYPosition = Double.parseDouble(prompt);
        }
        catch(NumberFormatException e) {
            JOptionPane.showMessageDialog(this, "Error, please input the desired initial Y Position [-1.0, 1.0]");
            String prompt = JOptionPane.showInputDialog("Please enter the desired initial Y Position [-1.0, 1.0]");
            promptYPosition = Double.parseDouble(prompt);
        }
        return promptYPosition;
    }


    //  Prompts the user for the turtle's initial state - Bearing
    private double userBearings() {
        double promptBearings = 0;
        try {
            String prompt = JOptionPane.showInputDialog("Please input the desired bearing you would like");
            promptBearings = Double.parseDouble(prompt);
        }
        catch(NumberFormatException e) {
            JOptionPane.showMessageDialog(this, "Error, please input the desired bearing you would like");
            String prompt = JOptionPane.showInputDialog("Please enter the desired bearing you would like");
            promptBearings = Double.parseDouble(prompt);
        }
        return promptBearings;
    }


    //  Prompts the user for the base segment length [1.0 means 1/2 image height]
    private double userBaseSegment() {
        double promptBaseSegment = 0;
        try {
            String prompt = JOptionPane.showInputDialog("Please input the desired base segment length you would like");
            promptBaseSegment = Double.parseDouble(prompt);
        }
        catch(NumberFormatException e) {
            JOptionPane.showMessageDialog(this, "Error, please input the desired base segment length you would like ");
            String prompt = JOptionPane.showInputDialog("Please enter the desired base segment length");
            promptBaseSegment = Double.parseDouble(prompt);
        }
        return promptBaseSegment;
    }


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


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


        // Sets the Background and the foreground colors
        Color color = new Color(newBackgroundColor);
        g2.setColor(color);
        g2.fill(new Rectangle(0,0,width,height));
        Color secondColor = new Color(newForegroundColor);
        g2.setColor(secondColor);


    }
    // Displays the Image after Configuration
    // Prompts the user for the number of generations, n
    // Prompts the user for the turtle's initial state [x,y, & bearing]: use relative coordinates [-1.0,1.0] for x and y
    // Where (0,0) represents the image's center - and the bearing is specified in degree with 0 degrees representing face right
    // (looking down the positive x-axis)
    // Prompts the user for the base segment length [1.0 means 1/2 the image's height]
    // Computes the n-th generation string as discussed in class
    // Visually interprets the n-gen string on the current image as discussed in class [the length of each will be (base seg length)/ ((scaling factor)^n)]
    // Displays the image


    public void displayTheImage()
    {
        // Calls the method that prompt for the user's desired number of generations
        int numOfGenerations = userGenerations();
        // Calls the method that prompt for the user's desired turtle X Position
        double turtleXPosition = userXPosition();
        // Calls the method that prompt for the user's desired turtle Y Position
        double turtleYPosition = userYPosition();
        // Calls the method that prompt for the user's desired bearing
        double bearing = userBearings();
        // Calls the method that prompt for the user's desired base segment length
```

```
double baseSegment = userBaseSegment();


baseSegment = baseSegment * this.height/2;

bearing = bearing * (Math.PI/180.0);

double newDelta = inputDelta * (Math.PI/180.0);

double newLength = baseSegment/(Math.pow(inputScaling, numOfGenerations));

turtleXPosition = (1 + turtleXPosition) * this.width/2;

turtleYPosition = (1 + turtleYPosition) * this.height/2;

double tempX = 0, tempY = 0;


String check = "";

if (numOfGenerations == 0)

{

    check = inputInitiator;

}

else

{

    check = stringGenerator(numOfGenerations);

}


// Moving the turtle as discussed in class


Stack<Point2D.Double> pointer = new Stack<Point2D.Double>();

Stack<Double> angle = new Stack<Double>();

BufferedReader reader = new BufferedReader(new StringReader(check));
```

```
        Line2D.Double line = new Line2D.Double(turtleXPosition, turtleYPosition,
turtleXPosition, turtleYPosition);
        int temp = 0;
        while(temp != -1)
        {
            try
            {
                temp = reader.read();
            }catch(IOException e){}
            if((char)temp == 'F')
            {
                line.setLine(turtleXPosition, turtleYPosition, turtleXPosition += newLength *
Math.cos(bearing), turtleYPosition -= newLength * Math.sin(bearing));
                g2.draw(line);
            }
            else if((char)temp == 'f')
            {
                turtleXPosition += newLength * Math.cos(bearing);
                turtleYPosition -= newLength * Math.sin(bearing);
            }
            else if((char)temp == '-')
            {
                bearing -= newDelta;
            }
            else if((char)temp == '+')
            {
```

```
            bearing += newDelta;
        }
        else if((char)temp == '[')
        {
            pointer.push(new Point2D.Double(turtleXPosition, turtleYPosition));
            angle.push(bearing);
        }
        else if((char)temp == ']')
        {
            Point2D.Double point = pointer.peek();
            pointer.pop();
            turtleXPosition = point.x;
            turtleYPosition = point.y;

            bearing = angle.peek();
            angle.pop();
        }
        else if((char)temp == 'R')
        {
            line.setLine(turtleXPosition, turtleYPosition, turtleXPosition += newLength * Math.cos(bearing), turtleYPosition -= newLength * Math.sin(bearing));
            g2.draw(line);
            bearing -= inputDelta;
            line.setLine(turtleXPosition, turtleYPosition, turtleXPosition += newLength * Math.cos(bearing), turtleYPosition -= newLength * Math.sin(bearing));
            g2.draw(line);
```

```
                bearing += (2 * inputDelta);

                line.setLine(turtleXPosition, turtleYPosition, turtleXPosition += newLength *
Math.cos(bearing), turtleYPosition -= newLength * Math.sin(bearing));

                g2.draw(line);

                bearing -= inputDelta;

                line.setLine(turtleXPosition, turtleYPosition, turtleXPosition += newLength *
Math.cos(bearing), turtleYPosition -= newLength * Math.sin(bearing));

                g2.draw(line);

            }

            else if((char)temp == 'r')

            {

                turtleXPosition += newLength * Math.cos(bearing);

                turtleYPosition -= newLength * Math.sin(bearing);

                bearing -= inputDelta;

                turtleXPosition += newLength * Math.cos(bearing);

                turtleYPosition -= newLength * Math.sin(bearing);

                bearing += (2 * inputDelta);

                turtleXPosition += newLength * Math.cos(bearing);

                turtleYPosition -= newLength * Math.sin(bearing);

                bearing -= inputDelta;

                turtleXPosition += newLength * Math.cos(bearing);

                turtleYPosition -= newLength * Math.sin(bearing);


            }

            else if((char)temp == 'L')

            {
                line.setLine(turtleXPosition, turtleYPosition, turtleXPosition += newLength *
Math.cos(bearing), turtleYPosition -= newLength * Math.sin(bearing));

                g2.draw(line);

                bearing += inputDelta;

                line.setLine(turtleXPosition, turtleYPosition, turtleXPosition += newLength *
Math.cos(bearing), turtleYPosition -= newLength * Math.sin(bearing));

                g2.draw(line);

                bearing -= (2 * inputDelta);

                line.setLine(turtleXPosition, turtleYPosition, turtleXPosition += newLength *
Math.cos(bearing), turtleYPosition -= newLength * Math.sin(bearing));

                g2.draw(line);

                bearing += inputDelta;

                line.setLine(turtleXPosition, turtleYPosition, turtleXPosition += newLength *
Math.cos(bearing), turtleYPosition -= newLength * Math.sin(bearing));

                g2.draw(line);

            }

            else if((char)temp == 'l')

            {

                turtleXPosition += newLength * Math.cos(bearing);

                turtleYPosition -= newLength * Math.sin(bearing);

                bearing += inputDelta;

                turtleXPosition += newLength * Math.cos(bearing);

                turtleYPosition -= newLength * Math.sin(bearing);

                bearing -= (2 * inputDelta);

                turtleXPosition += newLength * Math.cos(bearing);

                turtleYPosition -= newLength * Math.sin(bearing);
            bearing += inputDelta;
            turtleXPosition += newLength * Math.cos(bearing);
            turtleYPosition -= newLength * Math.sin(bearing);
        }
    }
    displayBufferedImage(image);
}

private String stringGenerator(int m){
    BufferedReader reader = null;
    char[] dictionary = new char[LSinstruct.length];
    String[] tempRules = new String[LSinstruct.length];
    int keyIndex = 0;
    String output = "";
    int temp = 0;
    for(int i = 0; i < LSinstruct.length; i++)
    {
        reader = new BufferedReader(new StringReader(LSinstruct[i]));
        try
        {
            dictionary[i] = (char)reader.read();
            tempRules[i] = LSinstruct[i].substring(4, LSinstruct[i].length());
        }catch(IOException e){}
    }
    reader = new BufferedReader(new StringReader(inputInitiator));
```

```
for(int i = 0; i < m; i++)
{
    String tempOutput = "";
    while(temp != -1)
    {
        try{
            temp = reader.read();
        }catch(IOException e){}
        for(int j = 0; j < dictionary.length; j++)
        {
            if((char)temp == dictionary[j])
            {
                keyIndex = j;
                break;
            }
            else keyIndex = -1;
        }
        if(keyIndex != -1)
            tempOutput += tempRules[keyIndex];
        else if(temp != -1)
            tempOutput += (char)temp;
    }
    temp = 0;
    reader = new BufferedReader(new StringReader(tempOutput));
    output = tempOutput;
```

```
        }
        return output;
    }


    //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();


    }


    // When the Save image options is selected, the program shall prompt the user for
the output file and save the current
    // L-system image as a PNG file


    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 );
    }
  }


}
```