Cover Page

I, \_*Xiaoxi Zheng*\_\_\_ affirm that the work submitted is my own and that the Honor  
Code was neither bent nor broken.

The easiest part of this HW is the setup of this project, since the structure of code  
was quite straight forward. The more difficult parts of the HW is buried  
within implementation of the algorithm, and the exact way the complex numbers are incremented. I also spend times debugging for Array out of bound exceptions when trying to convert coordinates back and forth between the Cartesian plane and the complex plane.

I believe the objective of this assignment was for us to understand and implement iterative patterns with the Mandelbrot and Julia set. These set memberships provide a visual on how complex numbers are represented. I also decided to implement the bonus part of this hw, so when the image zoomed, the aspect ratio stays constant.

Code

import java.util.Random;

import java.awt.Color;

import java.awt.\*;

import java.awt.event.\*;

import java.awt.image.\*;

import java.io.\*;

import javax.imageio.\*;

import javax.swing.\*;

import javax.swing.ImageIcon;

import javax.swing.JOptionPane;

import java.lang.Math.\*;

import java.lang.Math;

import java.awt.Graphics;

import java.awt.Graphics2D;

import java.awt.geom.\*;

import java.awt.geom.Line2D;

import javax.swing.JLabel;

import javax.swing.JMenuBar;

import javax.swing.JMenu;

import javax.swing.JMenuItem;

import java.lang.Math;

import java.lang.Character;

import java.util.Stack;

import java.util.Scanner;

import java.io.BufferedReader;

public class hw09{

private static final int WIDTH = 600;

private static final int HEIGHT = 450;

public static void main( String[] args){

SwingUtilities.invokeLater(new Runnable() {

public void run() {

createGUI();

}

});

}

private static void createGUI() {

JFrame frame = new ImageFrame(WIDTH,HEIGHT);

frame.setDefaultCloseOperation( JFrame.EXIT\_ON\_CLOSE);

frame.setVisible(true);

}

}

//####################################################################

class ImageFrame extends JFrame {

//private static final int INFINITE = 1000;

private AreaSelectPanel panel;

private JButton button;

private int width = 600;

private int height = 450;

private double x = 0;

private double y = 0;

//threashold for divergence test

private int tmax = 100;

private boolean mandelbrot;

private BufferedImage image = null;

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

private double r0 = -2;

private double r1 = 2;

private double deltaR = r1-r0;

private double i0 = -1.5;

private double i1 = 1.5;

private double deltaI = i1-i0;

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

//=========================

public ImageFrame(int width, int height){

this.setTitle("CAP 3027 2015 - HW09 -XiaoxiZheng");

this.setSize( width, height );

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

image = simulatedImage(width,height);

panel = new AreaSelectPanel( image);

button = new JButton( "Zoom" );

button.addActionListener( new ActionListener()

{

public void actionPerformed( ActionEvent event )

{

updateImage(mandelbrot);

System.out.println("Updating image");

}

} );

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

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

this.pack();

this.setVisible( true );

//this.setContentPane(new JScrollPane(label));

}

private void addMenu(){

JMenu fileMenu = new JMenu("File Menu");

//load IFS description

JMenuItem mandelbrot = new JMenuItem("Mandelbrot");

mandelbrot.addActionListener( new ActionListener(){

public void actionPerformed( ActionEvent event){

//initial μ value @(-2 + 1.5i) ---> Top Left

r0 = -2;

r1 = 2;

deltaR = r1-r0;

i0 = -1.5;

i1 = 1.5;

deltaI = i1-i0;

mandelbrot(r0,i0,r1,i1);

}

} );

fileMenu.add(mandelbrot);

JMenuItem julia = new JMenuItem("Julia Set");

julia.addActionListener( new ActionListener(){

public void actionPerformed(ActionEvent event){

//initial Z value @(-2 + 1.5i) ---> Top Left

constant = promptForMiu();

r0 = -2;

r1 = 2;

deltaR = r1-r0;

i0 = -1.5;

i1 = 1.5;

deltaI = i1-i0;

julia(r0,i0,r1,i1, constant[0], constant[1]);

}

} );

fileMenu.add(julia);

//Save image

JMenuItem saveImage = new JMenuItem("Save Image");

saveImage.addActionListener( new ActionListener(){

public void actionPerformed( ActionEvent event){

saveImage();

}

} );

fileMenu.add(saveImage);

//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);

}

private void mandelbrot(double ru, double iu,double ru1,double iu1){

//interpolate colors and store them in ColorSchema [] array

interpolateColor();

deltaR = ru1- ru;

deltaI = iu1 - iu;

double realIncr = (ru1- ru)/(width - 1);

double imagIncr = (iu1 - iu)/(height -1);

//initial μ value @(-2 + 1.5i) ---> Top Left

double real = ru;

//mandelbrot algorithm

//looping thru a 600\*450 bounded region

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

double img = iu1;

for(int y=0; y<height; y++){

double [] complexZ = new double [2];

// temp variable to store real and img part of z when computing;

complexZ[0] = 0;

complexZ[1] = 0;

int t = 0;

while(t!=tmax){

//z = (z\*z) + u

complexZ = zSquarePlusMiu(complexZ[0],complexZ[1],real,img);

if(sumOfSquare(complexZ[0],complexZ[1]) > 4.0) {

break;//diverging

}

else{

++t;

}

}

if(t == tmax){

//Plot black

double [] bitmapCoord = new double [2];

//temp variable to store the converted bitmap //interpretation of complex number

bitmapCoord = toBitmapCoord(real,r0,r1,img,i0,i1);

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

}

else{

//diverged and μ is not in Mandelbrot set

//plot μ using colorSchema[t].

double [] bitmapCoord = new double [2];

bitmapCoord = toBitmapCoord(real,r0,r1,img,i0,i1); image.setRGB((int)(bitmapCoord[0]),(int)(bitmapCoord[1]),colorSchema[t]);

}

img -= imagIncr;

}

real += realIncr;

}

//updateImage();

SwingUtilities.invokeLater(new Runnable() {

public void run() {

//displayFile(image);

mandelbrot = true;

panel.setImage(image);

}

});

}

private void julia(double rz, double iz, double rz1, double iz1, double rMiu\_, double iMiu\_){

double [] complexU = new double [2];

complexU[0] = rMiu\_;

complexU[1] = iMiu\_;

//interpolate colors and store them in ColorSchema [] array

interpolateColor();

deltaR = rz1- rz;

deltaI = iz1 - iz;

double realIncr = (rz1- rz)/(width - 1);

double imagIncr = (iz1 - iz)/(height -1);

//initial μ value @(-2 + 1.5i) ---> Top Left

double real = rz;

//imgU = iu;

//Julia algorithm

//looping thru a 600\*450 bounded region

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

double img = iz1;

//realU = realU + 4/width;

for(int y=0; y<height; y++){

double [] complexZ = new double [2];

complexZ[0] = real;

complexZ[1] = img;

int t = 0;

while(t!=tmax){

//z = (z\*z) + u

complexZ = zSquarePlusMiu(complexZ[0],complexZ[1],complexU[0],complexU[1]);

if(sumOfSquare(complexZ[0],complexZ[1]) > 4.0) {

break;//diverging

}

else{

++t;

}

}

if(t ==tmax){

//Plot black

double [] bitmapCoord = new double [2];

bitmapCoord = toBitmapCoord(real,r0,r1,img,i0,i1);

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

}

else{

//diverged and μ is not in Mandelbrot set

//plot μ using colorSchema[t].

double [] bitmapCoord = new double [2]; //temp variable to store the converted bitmap interpretation of complex number

bitmapCoord = toBitmapCoord(real,r0,r1,img,i0,i1);

image.setRGB((int)(bitmapCoord[0]),(int)(bitmapCoord[1]),colorSchema[t]);

}

img -= imagIncr;

}

real += realIncr;

}

SwingUtilities.invokeLater(new Runnable() {

public void run() {

//displayFile(image);

mandelbrot = false;

panel.setImage(image);

}

});

}

private double [] zSquarePlusMiu(double rZ,double iZ,double rU,double iU){

double [] answer = new double [2];

//compute Z^2 + μ in complex form

answer[0] = (rZ\*rZ) - (iZ\*iZ)+ rU;

//answer[1] = ((rZ\*iZ)+ (iZ\*rZ)) + iU;

answer[1] = (2\*rZ\*iZ) + iU;

return answer;

}

private double sumOfSquare(double realZ\_,double imgZ\_){

double answer = 0.0;

answer = realZ\_\*realZ\_ + imgZ\_\*imgZ\_;

return answer;

}

private double [] toBitmapCoord(double realU\_,double r0\_,double r1\_,double imgU\_,double i0\_,double i1\_){

double answer [] = new double [2];

double deltaR\_ = r1\_-r0\_;

double deltaI\_ = i1\_-i0\_;

answer[0] = ((realU\_ - r0\_ ) / deltaR\_\* (width-1));

answer[1] = (imgU\_ - i0\_ ) / deltaI\_ \* (height-1);

return answer;

}

private void interpolateColor(){

int ARGBNewGeneral = 0;

double[] colorInfoLeft;

double[] colorInfoRight;

//ignores delta alpha bc in this hw has no changes in alpha

double deltaRGen;

double deltaGGen;

double deltaBGen;

double redGeneral; //start paiting @ left

double greenGeneral; //start paiting @ left

double blueGeneral; //starting paiting @ left

//color[0] = white, color[40] = red, color[100] = Blue

//interpolate from 0---50

//follows the rule of thumb of right hand side of canvas - left hand side of canvas

colorInfoLeft = extraction(16711680);//extract white

colorInfoRight = extraction(16747520);//extract orange

//ignores delta alpha bc in this hw has no changes in alpha

deltaRGen = (colorInfoRight[1] - colorInfoLeft[1])/(49); //[1]--channel for red

deltaGGen = (colorInfoRight[2] - colorInfoLeft[2])/(49); //[2]--channel for green

deltaBGen = (colorInfoRight[3] - colorInfoLeft[3])/(49); //[3]--channel for blue

redGeneral = colorInfoLeft[1]; //start paiting @ left

greenGeneral = colorInfoLeft[2]; //start paiting @ left

blueGeneral = colorInfoLeft[3]; //starting paiting @ left

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

redGeneral = redGeneral + deltaRGen;

//bc red starts from the left

greenGeneral = greenGeneral + deltaGGen;

blueGeneral = blueGeneral + deltaBGen;

//clamping

if(redGeneral>255){

redGeneral = 255;

}

if(redGeneral<0){

redGeneral = 0;

}

if(greenGeneral>255){

greenGeneral = 255;

}

if(greenGeneral<0){

greenGeneral = 0;

}

if(blueGeneral>255){

blueGeneral = 255;

}

if(blueGeneral<0){

blueGeneral = 0;

}

ARGBNewGeneral = toIntARGB(255,redGeneral,greenGeneral,blueGeneral);

colorSchema[x] =ARGBNewGeneral

}

//50---100

//follows the rule of thumb of right hand side of canvas - left hand side of canvas

colorInfoLeft = extraction(16747520);//extract orange

colorInfoRight = extraction(16711680);//extract red

//ignores delta alpha bc in this hw has no changes in alpha

deltaRGen = (colorInfoRight[1] - colorInfoLeft[1])/(49); //[1]--channel for red

deltaGGen = (colorInfoRight[2] - colorInfoLeft[2])/(49); //[2]--channel for green

deltaBGen = (colorInfoRight[3] - colorInfoLeft[3])/(49); //[3]--channel for blue

redGeneral = colorInfoLeft[1]; //start paiting @ left

greenGeneral = colorInfoLeft[2]; //start paiting @ left

blueGeneral = colorInfoLeft[3]; //starting paiting @ left

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

redGeneral = redGeneral + deltaRGen; //bc red starts from the left

greenGeneral = greenGeneral + deltaGGen;

blueGeneral = blueGeneral + deltaBGen;

//clamping

if(redGeneral>255){

redGeneral = 255;

}

if(redGeneral<0){

redGeneral = 0;

}

if(greenGeneral>255){

greenGeneral = 255;

}

if(greenGeneral<0){

greenGeneral = 0;

}

if(blueGeneral>255){

blueGeneral = 255;

}

if(blueGeneral<0){

blueGeneral = 0;

}

ARGBNewGeneral = toIntARGB(255,redGeneral,greenGeneral,blueGeneral);

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

}

}

private static double[] extraction(int ARGB\_){

double[] extractionArray;

extractionArray = new double[4];

//extractionArray -- extraction[0] = alpha values;

//extraction[1] = red values; & etc with ARGB

extractionArray[0] = ARGB\_>>>24;

extractionArray[1] = (ARGB\_<<8) >>> 24;

extractionArray[2] = (ARGB\_<<16)>>>24;

extractionArray[3] = (ARGB\_<<24)>>>24;

return (extractionArray);

}

private int toIntARGB(double alpha\_, double red\_, double green\_, double blue\_){

//System.out.println((alpha\_<<24)|(red\_<<16)|(green\_<<8)|(blue\_));

return ((((int)alpha\_)<<24)|(((int)red\_)<<16)|((int)(green\_)<<8)|((int)blue\_));

}

private void saveImage(){

try

{

File outputfile = new File("IFS.png");

javax.imageio.ImageIO.write(image, "png", outputfile );

}

catch ( IOException e )

{

JOptionPane.showMessageDialog( ImageFrame.this,

"Error saving file",

"oops!",

JOptionPane.ERROR\_MESSAGE );

}

}

private double [] promptForMiu(){

double [] temp = new double[2];

double [] error = new double[2];

error[0] = -100000;

error[1] = 100000;

String input1 = JOptionPane.showInputDialog("Please enter the real part of Mu ");

String input2 = JOptionPane.showInputDialog("Please enter the imaginary part of Mu");

if(valideInput(input1) && valideInput(input2)){

temp[0] = Double.parseDouble(input1);

temp[1] = Double.parseDouble(input2);

return temp;

}

else if (input1 == null || input2 == null){ //User clicked "Cancel"

System.exit(0);

return error;

}

else{

return promptForMiu();

}

}

private boolean valideInput(String input\_){

try{

double num = Double.parseDouble(input\_);

if(num<-1000 || num > 1000){

JOptionPane.showMessageDialog(null, "Invalid Input", "alert", JOptionPane.ERROR\_MESSAGE);

return false;

}

return true;

}

catch(NumberFormatException e){

JOptionPane.showMessageDialog(null, "Invalid Input", "alert", JOptionPane.ERROR\_MESSAGE);

return false;

}

}

protected BufferedImage simulatedImage(int width\_,int height\_){

while (true) {

if (width\_ < 0 || height\_ < 0)

return null;

try {

BufferedImage img = new BufferedImage(width\_,height\_,BufferedImage.TYPE\_INT\_RGB);

return img;

} catch (OutOfMemoryError err) {

JOptionPane.showMessageDialog(this, "Ran out of memory! Try using a smaller image size.");

}

}

}

public void updateImage(boolean mandelbrot\_){

double new\_r0 = panel.getUpperLeft().getX() \* deltaR + r0;

double new\_r1 = panel.getLowerRight().getX()\* deltaR + r0;

double new\_i0 = panel.getUpperLeft().getY() \* deltaI + i0;

double new\_i1 = panel.getLowerRight().getY() \* deltaI + i0;

r0 = new\_r0;

r1 = new\_r1;

i0 = new\_i0;

i1 = new\_i1;

if(mandelbrot\_){

//call mandelbrot

mandelbrot(r0,i0,r1,i1);

}

else{

//call Julia

julia(r0, i0, r1, i1,constant[0], constant[1]);

}

}

}

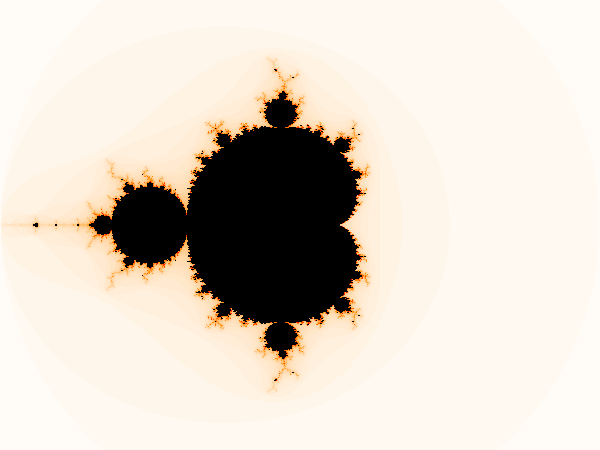
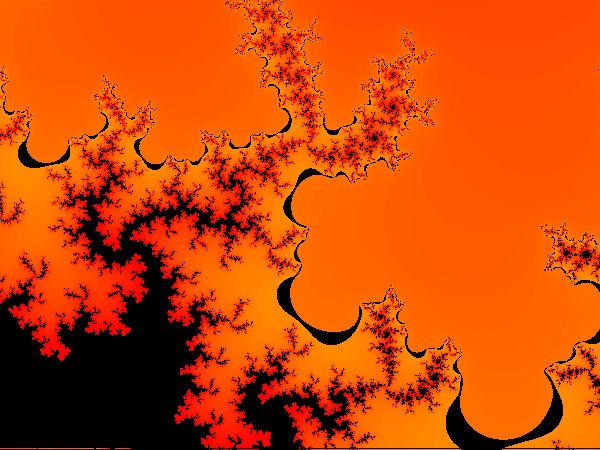
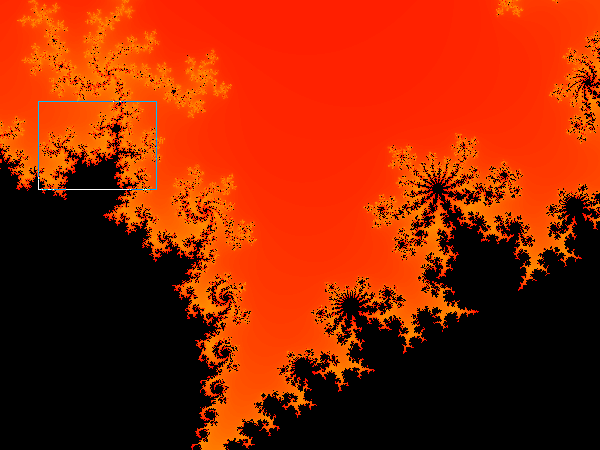
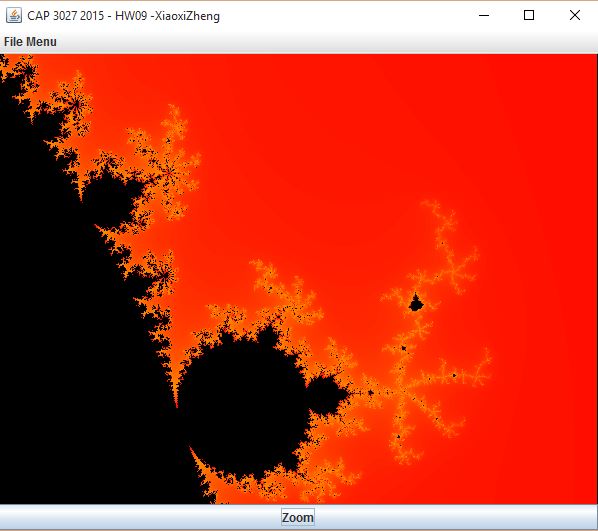
=====================================================================================

Question

1. Does the program compile without errors?  
Yes.  
2. 2. Does the program compile without warnings?  
Yes  
3. 3. Does the program run without crashing?  
Yes  
4. 4. Describe how you tested the program.  
I ran several test cases with different user inputs. I gave couple illegal test cases and I also zoomed in a number of times to make sure my program won’t crash.   
5. Describe the ways in which the program does not meet assignment's specifications.  
None.  
6. Describe all known and suspected bugs.  
There are no known bugs.  
7. Does the program run correctly?  
Yes

Screenshots

## Mandelbrot set



## Julia Set

