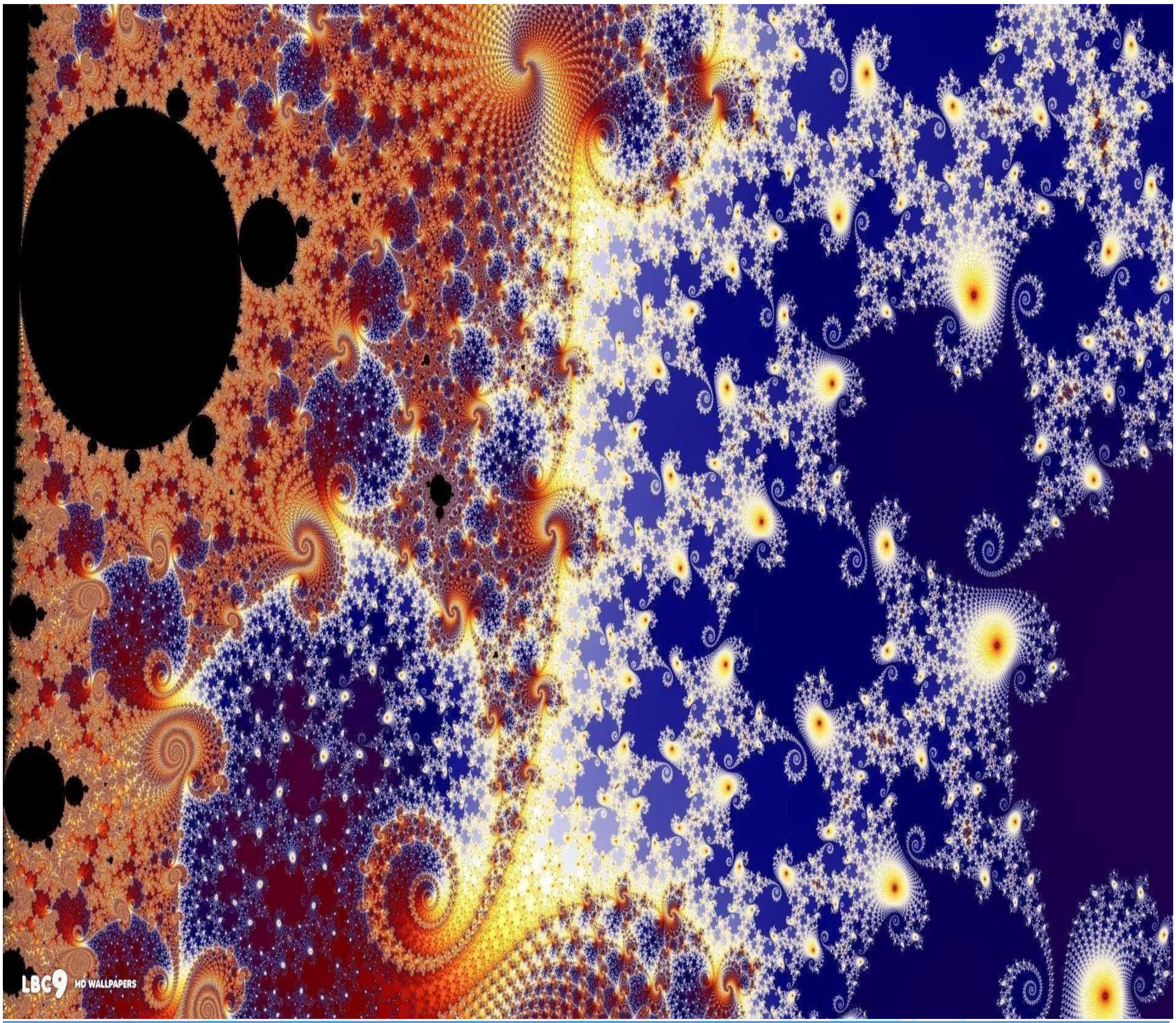


Advance Programming Project – Mandelbrot and Julia Set with Java

Professor : Dr.Zahra Movahedi
Collegians : Ahmad Reza Parsi Zadeh
Gmail : ahmad.parsizaseh@gmail.com
University : University of Tehran

In the name of God

Hi

First I have to give you an overview of the project.

This project has a mathematical basis and a mathematical concept has been used.

So first we will explain the Mandelbrot and Julia Set and explain its logic to you, then we will explain a series of basic concepts to make the project easier for you to understand.

Finally, we will explain the algorithm, functions and methods, as well as attributes. Finally, in this file, you can see the diagram file of this project.

The Mandelbrot set is a set of Complex points on the coordinate plane

To build the Mandelbrot set, we must use an algorithm based on the recursive formula:

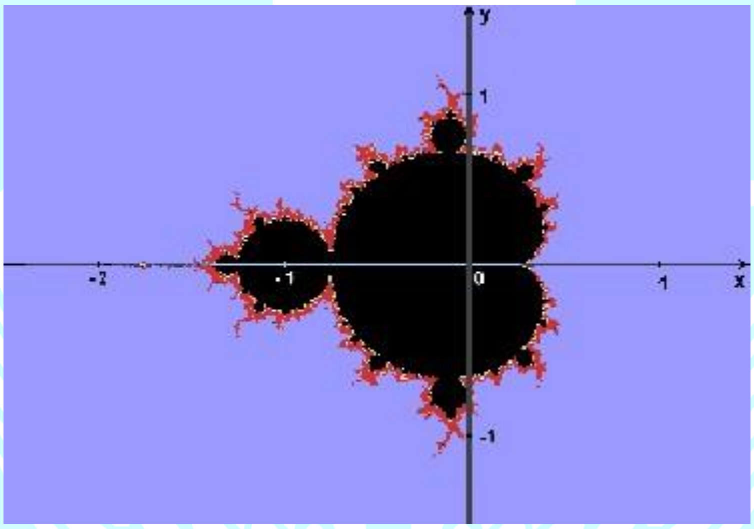
$$Z_n=(Z_{n-1})^2+C$$

Page points are divided into two categories:

1- Points within the Mandelbrot collection

2 – Points Outside the Mandelbrot collection

A color can be assigned to points outside the Mandelbrot collection, the color of which depends on the number of repetitions to determine points outside the Mandelbrot collection.



Creating the Mandelbrot Set :

To form this set, we must select a point that represents a complex number :

$$C=a+b.i$$

After Recursive calculating the $Z_1=(Z_0)^2+C$ and putting a value of zero for the Z_0, the value of C is obtained as a result.

Finally, we go through this return process.

The interesting thing about this shape is that if we do not set a limit for the completion of these shapes, this shape (Mendelbrot set) will increase infinitely, while around this shape where the Julia set occurs, as explained above. Given will decrease with infinite speed and will have internal growth.

This was a series of descriptions of these collections to better understand the code related to this project.

The code starts with some necessary libraries which are imported.

A short explanation of each of them could be helpful.

```
import java.awt.*;
```

It is *an API to develop GUI or window-based applications* in java.

Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavyweight i.e. its components are using the resources of OS.

```
import javax.swing.*;
```

It is a part of Java Foundation Classes (JFC) that is *used to create window-based applications*. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.

Unlike AWT, Java Swing provides platform-independent and lightweight components.

So , it was a brief explanation of the libraries and In the following, we will deal with the contents of each class and the function of each part of the code…

First of all we have some attributes that are necessary and I will explain every usage of them in the project.

The constructor of the main class contains some methods , first method is **setInitialGUIProperties().** in this method we are initializing important things to create the base of our project.

```
private void setInitialGUIProperties() {
/*here it just creates a simple page*/
    this.setTitle("Fractal Explorer");//It is setting the title at the top of
the stage page
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);//The window close
naturally by clicking one the exit button
    this.setSize(WIDTH, HEIGHT);//It is setting the size of the stage page
this.setResizable(false);//This makes the window without resizablity because
we don't wanna to do that!
    this.setLocationRelativeTo(null);//make sure that when our GUI is created
it will show up at the center of the screen
} // setInitialGUIProperties
```

at the first line we are setting our page's title. The second line says that the window closes automatically when the user logs out and after that I set the frame or size of the application page With a specified length and width.

If you pass a false logic variable as the argument of the next function then you can't change the size of the page with any buttons or anything else. And when you call the last function in fact you are making sure that your page will appear in the middle of your screen.

```
private class Canvas extends JPanel implements MouseListener
```

I had to create a class named **Canvas** which extends (inherits) the JPanel and impelements MousListener interface.

JPanel is a part of Java Swing package, is a container that can store a group of components. The main task of JPanel is to organize components, various layouts can be set in JPanel which provide better organisation of components.

MouseListener and MouseMotionListener is an interface in java.awt.event package .In fact this class and the methods which are in it handling the actions like right click , left click and something like this.

```
@Override public Dimension getPreferredSize() {
    return new Dimension(WIDTH, HEIGHT);//it returns our preffered size of
canvas
} // getPreferredSize
```

This function is an overrided function which it's return type is **Dimension** well this type is setting the size that you want for your Canvas

In addCanvas method we have a line of code which used **BufferedImage** class.

Java BufferedImage class is a subclass of Image class. It is used to handle and manipulate the image data. A BufferedImage is made of ColorModel of image data. All BufferedImage objects have an upper left corner coordinate of (0, 0).

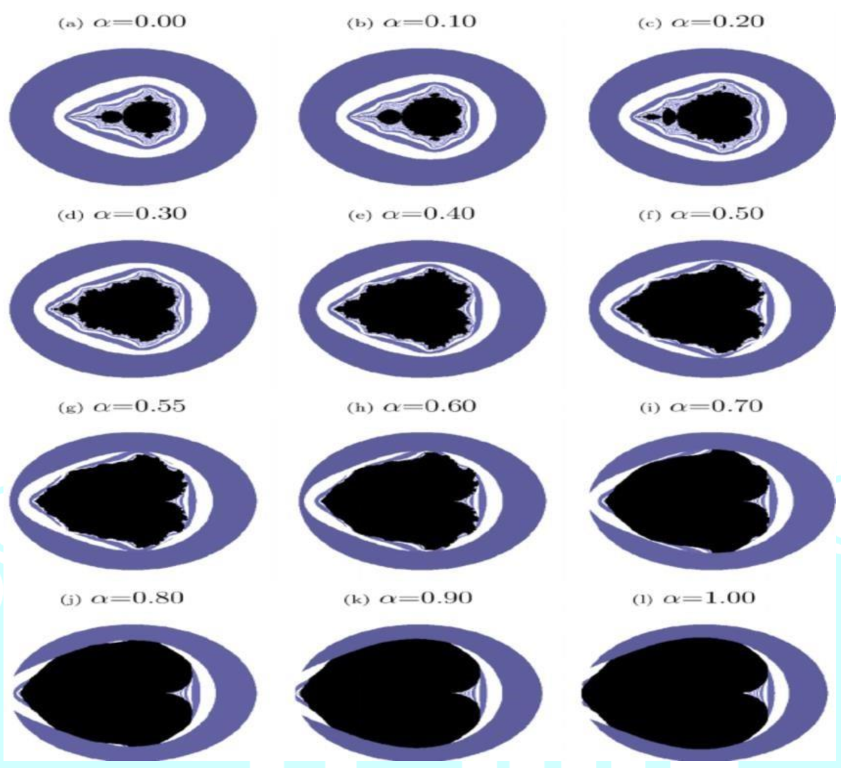
And two last methods in that function are so easy. One of them making the canvas visible and the other add it to the canvas but the second argument has BufferedLayout type and this class will give you a temporary memory and after that you will apply some changes but it wont be appear in the screen but when you call applyLayout ou add it to your canvas it will be visible.

The next function that I want to explain is **computeItrations:**

```
private int computeIterations(double c_r, double c_i) {
      //as arguments it will take two part of the complex number first is
real part an second is the imaginary part
/*
Let c = c_r + c_i
Let z = z_r + z_i
z' = z*z + c
   = (z_r + z_i)(z_r + z_i) + (c_r + c_i)
   = z_r² + 2*z_r*z_i - z_i² + c_r +
c_i      z_r' = z_r² - z_i² + c_r
z_i' = 2*z_i*z_r + c_i
*/          double z_r =
0.0;         double z_i =
0.0;         int
iterCount = 0;

      // Modulus (distance) formula:
      // √(a² + b²) <= 2.0
      // a² + b² <= 4.0
      while ( z_r*z_r + z_i*z_i <= 4.0 ) {
           double z_r_tmp =
z_r;
           z_r = z_r*z_r - z_i*z_i +
c_r;           z_i = 2*z_i*z_r_tmp +
c_i;

          // Point was inside the Mandelbrot
set            if (iterCount >= MAX_ITER)
return MAX_ITER;

iterCount++;

      }
      return iterCount;}
```

This method takes two parameters that actually provide a complex number. first argument is the real part and second one is the imaginary part of our complex number. The explanation of this method is a little complicated but

I am going to explain it perfect.there is an image that will explain it obviously.



In this image you will see that in the beginning it was a circle and this circle will change to the shape that we want. First comments described the formula and the logic of this method if you see I initialized the imaginary and the real part of z with zero but why the condition of the while loop is like that ? because that is our first circle which changes it's shape into the Mandelbrot set.

So we have the formula , we have our two complex numbers , and finally in the loop we have a temp number for calculating the set correctly with our formula.

However if the while loop breaks out , it means our complex number we chose is outside the set and it will break the loop. But if it doesn't happen for every iteration it returns the maximum number of iteration.(MAX_ITER)

Well , one of the most important methods is **updateFractal().**

```
/**
*     Updates the fractal by computing the number of iterations   *
for each point in the fractal and changing the color
*     based on that.
 **/
public void updateFractal() {
     for (int x = 0; x < WIDTH; x++ ) {
for (int y = 0; y < HEIGHT; y++ ) {
             double c_r =
getXPos(x);                double c_i
= getYPos(y);

            int iterCount = computeIterations(c_r, c_i);
             int pixelColor =
makeColor(iterCount);
fractalImage.setRGB(x, y, pixelColor);


        }
}
    canvas.repaint();}
```

Before describing this method I am going to show you some other attributes.

```
static final double DEFAULT_ZOOM       = 100.0;
static final double DEFAULT_TOP_LEFT_X = -3.0;
static final double DEFAULT_TOP_LEFT_Y = +3.0;
double zoomFactor = DEFAULT_ZOOM; double
topLeftX   = DEFAULT_TOP_LEFT_X; double
topLeftY   = DEFAULT_TOP_LEFT_Y;
```

this attributes are used in method which I want to explain. DEFAULT_ZOOM is showing that how depth we are in the pricture. This method will going through every single picture on our canvas and changes its color.

With x , y we can get every single pixel on our canvas.

There is two other methods :

```
private double getXPos(double x) {//it is going to transform the x
coordinates into a value on the complex plane or rather into a point
return x/zoomFactor + topLeftX;
} // getXPos
// ----------------------------------------------------------------------
private double getYPos(double y) {     return y/zoomFactor - topLeftY;
} // getYPos
```

our x acces is going to be the real part and this method as I sayd in the comments is going to transform a x coordinate into a value on our complex plane.after this we have to create our color based on the number of iteration.

```
/** Returns a posterized color based off of the iteration
count  of a given point in the fractal **/ private int
makeColor( int iterCount ) {
    int color = 0b011011000001100101101000;//it is a color that has a bit
red , green and blue . it is more blue and your can change it
    int mask  = 0b000000000000010101110111;//these two numbers have "0b"
that means they are binary numbers
    int shiftMag = iterCount / 13;//because in previous number we have 13
zeros at the beginning
     if (iterCount == MAX_ITER)
return Color.CYAN.getRGB();


    return color | (mask << shiftMag);//| means binary or
} // makeColor
/**
*     Notice :
*     if you use & operator it will draw an image that has black and a
*     lot of red and yellow and a bit green on the borders of the shape
(Horror!)
*     but if you use | operator it will be the same image as you saw  *
**/
```

You can recognize that inside of our Mandelbrot Set color is CYAN. It is my favorite color ☺. So otherwise you can see some numbers at the top of the function and some pieces of code that they need to be more clear but before that if you don't know to how zoom into the Canvas and neither zoom out , it doesn't make any sense . So…

Well we have to override some methods for zooming in or zooming out. As an argument I passed a MouseEvent object to the all of the mentioned functions which I want to override. First I have to find where the mouse is clicked. Actually I have to find the x and y coordinates of the mouse clicked.

After that we have to figure out with switch command that which one of the buttons are clicked ? right one or left one ? so to figuring out I used this part of code :

```
@Override public void mousePressed(MouseEvent mouse) {
    //getting the x coordinates and y coordinates that our mouse is
clicked     double x = (double) mouse.getX();     double y = (double)
mouse.getY();
    switch( mouse.getButton() ) {//this will determine that i pressed the
right mouse button or left mouse button
        // Left          case
MouseEvent.BUTTON1:
            //zooming in
            adjustZoom( x, y, zoomFactor*2 );
break;
        //BUTTON2 is reserved for some mouses that have the scroll ine
the middle         case MouseEvent.BUTTON3:
            adjustZoom( x, y, zoomFactor/2 );
break;}
}
```

If you look at your mouse there is three buttons , right button that BUTTON1 is reserved for it. And left button that BUTTON3 is reserved for it.

I know I have to declare some other methods that I didn't yet but before those let's see another function called :

```
private void adjustZoom( double newX, double newY, double newZoomFactor ) {
//these first two lines zoom into the fractal by shifting the topleft x and
toplefy
    topLeftX += newX/zoomFactor;
topLeftY -= newY/zoomFactor;
zoomFactor = newZoomFactor;
     topLeftX -= ( WIDTH/2) /
zoomFactor;      topLeftY += (HEIGHT/2)
/ zoomFactor;


    updateFractal();


} // adjustZoom
```

first two lines are shifting the picture and after we passed the newZoomFactor to the zoomFactor , because we want to recenter the picture after zooming in or zooming out I used two last lines of code.

If you look at makeColor method I used to binary numbers that show some color. This part of code is related to binary operations because there is a concept that it is called **mask** well I am not going to explain it but based on the iterCount which is its argument , I divided this number to 13 because for shifting we have 13 zeros at the beginning of the mask number and for each number we passed as an argument it will shift it to the mask.(Actually it is creating a mask for each time we call it in for loop.) and finally after that I used pip (binary OR operation). All pf these things that used is for creating a bunch of colors based binary numbers to create a shape with colors of rainbow. (a continuous range of colors)

There are three main and very important and difficult functions that I must explain the relationship between them in general.

The three updateFractal, makeColor and computeIteration, are very closely related.

The computeIteration function first determines whether the point we are looking for exists within the set or not.

If you look closely at the makeColor function, you'll notice that I used a conditional expression that says if the output number is equal to maxIter, set the color to sky blue and it changes the color of the pixel.

But in my opinion, the most important function is the updateFractal function, because first, to access the individual pixels in the image, we make a twodimensional loop and use the functions we used to convert these coordinates to a complex number. And then we use the computer function to find out if this point is inside our set or not, after which the output number of the computer function and the makeColor function and where the color is The basis of the position of the pixel is determined.

And finally we have some methods for key stroking :

```
// -----------------------------------------------------------------Done
private void moveUp() {//if your want to go up half the screen up then put
/2
    double curHeight = HEIGHT / zoomFactor;
topLeftY += curHeight / 6;
updateFractal();
} // moveUp
// -----------------------------------------------------------------Done
private void moveDown() {//if your want to go down half the screen down
then put /2
    double curHeight = HEIGHT / zoomFactor;
topLeftY -= curHeight / 6;
updateFractal();
} // moveDown
// -----------------------------------------------------------------Done
private void moveLeft() {//if your want to go left half the screen left
then put /2
    double curWidth = WIDTH / zoomFactor;
topLeftX -= curWidth / 6;
updateFractal();
} // moveLeft
// -----------------------------------------------------------------Done
private void moveRight() {//if your want to go right half the screen right
then put /2
    double curWidth = WIDTH / zoomFactor;
topLeftX += curWidth / 6;
updateFractal();
} // moveRight
```

Well I determined a variable called **curHeight** that determines where we actually are in the canvas and I calculated it and initialized it to this variable. After that if you want to go up fast , so you hav to divide this number to 2 or 3 that is completely optional. So these four methods are so simple ☺.

For going up , left and somethings like this I used some keyboardEvents and mapped them to an appropriate Action.

```
//to mapping the key to the event this.getInputMap().put( wKey,
"w_key" ); this.getInputMap().put( aKey, "a_key" );
this.getInputMap().put( sKey, "s_key" ); this.getInputMap().put(
dKey, "d_key" );


this.getActionMap().put( "w_key", wPressed );
```

```
this.getActionMap().put( "a_key", aPressed );
this.getActionMap().put( "s_key", sPressed );
this.getActionMap().put( "d_key", dPressed );
```

Here you can see they are so simple just like the previous method but :

```
Action wPressed = new AbstractAction() {
    @Override public void actionPerformed(ActionEvent e) {
moveUp();
    }
};


Action aPressed = new AbstractAction() {
    @Override public void actionPerformed(ActionEvent e) {
moveLeft();
    }
};


Action sPressed = new AbstractAction() {
    @Override public void actionPerformed(ActionEvent e) {
moveDown();
    }
};


Action dPressed = new AbstractAction() {
    @Override public void actionPerformed(ActionEvent e) {
moveRight();
    }
};
```

Here you can see some Abstract methods which determines for every keys events an appropriate action.

Finally, I hope you enjoyed this explanation and that it helped you understand this code. If you have any questions about this, I'll be happy for you to share them with me.

Enjoy programming

Sincerely