Министерство Цифрового Развития, Связи и Массовых Коммуникаций Российской Федерации Ордена Трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение высшего образования   
“Московский технический университет связи и информатики”

Кафедра «Информационные технологии»

Лабораторная работа №4:   
«Рисование фракталов**»**

Выполнил: студент гр. БПЗ1902

Кварацхелия Д.Т.

Проверил: Харрасов К.Р.

Москва 2021 г.

Цель работы:

В данной лабораторной работе необходимо создать небольшое JAVA-приложение, которое сможет рисовать фракталы.

Задание:

• Класс FractalExplorer должен отслеживать несколько важных полей для состояния программы:

1) Целое число «размер экрана», которое является шириной и высотой отображения в пикселях. (Отображение фрактала будет квадратным.)   
 2) Ссылка JImageDisplay, для обновления отображения в разных методах в процессе вычисления фрактала.   
 3) Объект FractalGenerator. Будет использоваться ссылка на базовый класс для отображения других видов фракталов в будущем.   
 4) Объект Rectangle2D.Double, указывающий диапазона комплексной плоскости, которая выводится на экран.

• У класса должен быть конструктор, который принимает значение размера отображения в качестве аргумента, затем сохраняет это значение в соответствующем поле, а также инициализирует объекты диапазона и фрактального генератора. Данный конструктор не должен устанавливать какие-либо компоненты Swing; они будут установлены в следующем методе.

• Создайте метод createAndShowGUI (), который инициализирует графический интерфейс Swing: JFrame, содержащий объект JimageDisplay, и кнопку для сброса отображения. Используйте java.awt.BorderLayout для содержимого окна; добавьте объект отображения изображения в позицию BorderLayout.CENTER и кнопку в позицию BorderLayout.SOUTH.

• Реализуйте вспомогательный метод с типом доступа private для вывода на экран фрактала, можете дать ему имя drawFractal ().

• Создайте внутренний класс для обработки событий java.awt.event.ActionListener от кнопки сброса. Обработчик должен сбросить диапазон к начальному, определенному генератором, а затем перерисовать фрактал.

• Создайте другой внутренний класс для обработки событий java.awt.event.MouseListener с дисплея. Вам необходимо обработать события от мыши, поэтому вы должны унаследовать этот внутренний класс от класса MouseAdapterAWT.

• В заключении, вам необходимо создать статический метод main() для FractalExplorer так, чтобы можно было его запустить.

Выполнение работы:

1. Создание пользовательского интерфейса:

Прежде чем рисовать фракталы, необходимо создать графический виджет, который будет их отображать.

• Создайте класс JImageDisplay. Класс BufferedImage управляет изображением, содержимое которого можно записать.

import javax.swing.JComponent;

import java.awt.\*;

import java.awt.image.BufferedImage;

public class JImageDisplay extends JComponent{

    private BufferedImage image;

    JImageDisplay(int w,int h){

        image = new BufferedImage(w,h, BufferedImage.TYPE\_INT\_RGB);

        Dimension dimension = new Dimension(w, h);

        super.setPreferredSize(dimension);

    }

    protected void paintComponent(Graphics g){

        super.paintComponent(g);

        g.drawImage (image, 0, 0, image.getWidth(), image.getHeight(), null);

    }

    public void clearImage(){

        int[] emptyArr= new int[getHeight()\*getWidth()];

        image.setRGB(0,0,getWidth(),getHeight(),emptyArr,0,1);

    }

    public void drawPixel(int x, int y, int rgbColor){

        image.setRGB(x,y,rgbColor);

    }

}

2. Код для вычисления фрактала Мандельброта.

Для создания фракталов используем следующий исходный файл FractalGenerator.java, от которого будут унаследованы все фрактальные генераторы.

import java.awt.geom.Rectangle2D;

/\*\*

 \* This class provides the common interface and operations for fractal

 \* generators that can be viewed in the Fractal Explorer.

 \*/

public abstract class FractalGenerator {

    /\*\*

     \* This static helper function takes an integer coordinate and converts it

     \* into a double-precision value corresponding to a specific range.  It is

     \* used to convert pixel coordinates into double-precision values for

     \* computing fractals, etc.

     \*

     \* @param rangeMin the minimum value of the floating-point range

     \* @param rangeMax the maximum value of the floating-point range

     \*

     \* @param size the size of the dimension that the pixel coordinate is from.

     \*        For example, this might be the image width, or the image height.

     \*

     \* @param coord the coordinate to compute the double-precision value for.

     \*        The coordinate should fall in the range [0, size].

     \*/

    public static double getCoord(double rangeMin, double rangeMax,

        int size, int coord) {

        assert size > 0;

        assert coord >= 0 && coord < size;

        double range = rangeMax - rangeMin;

        return rangeMin + (range \* (double) coord / (double) size);

    }

    /\*\*

     \* Sets the specified rectangle to contain the initial range suitable for

     \* the fractal being generated.

     \*/

getInitialRange (Rectangle2D.Double) - метод позволяет генератору фракталов определить наиболее «интересную» область комплексной плоскости для конкретного фрактала.

    public abstract void getInitialRange(Rectangle2D.Double range);

    /\*\*

     \* Updates the current range to be centered at the specified coordinates,

     \* and to be zoomed in or out by the specified scaling factor.

     \*/

    public void recenterAndZoomRange(Rectangle2D.Double range,

        double centerX, double centerY, double scale) {

        double newWidth = range.width \* scale;

        double newHeight = range.height \* scale;

        range.x = centerX - newWidth / 2;

        range.y = centerY - newHeight / 2;

        range.width = newWidth;

        range.height = newHeight;

    }

    /\*\*

     \* Given a coordinate <em>x</em> + <em>iy</em> in the complex plane,

     \* computes and returns the number of iterations before the fractal

     \* function escapes the bounding area for that point.  A point that

     \* doesn't escape before the iteration limit is reached is indicated

     \* with a result of -1.

     \*/

    public abstract int numIterations(double x, double y);

}

Метод numIterations(double, double) реализует итеративную функцию для фрактала Мандельброта

Создайте подкласс FractalGenerator с именем Mandelbrot. в нем вам необходимо будет обеспечить только два метода: getInitialRange() и numIterations().

import java.awt.geom.Rectangle2D;

public class Mandelbrot extends  FractalGenerator{

    public static final int MAX\_ITERATION = 2000;

    public void getInitialRange(Rectangle2D.Double range){

        range.x = -2;

        range.y = -1.5;

        range.width = 3;

        range.height = 3;

    }

    public int numIterations(double x, double y){

        int i =0;

        double zR=0;

        double zI =0;

        while (i < MAX\_ITERATION && zR \* zR + zI \*zI <4){

            double tempZR = zR \* zR -zI\*zI+x;

            double tempZI = 2\*zR\*zI+y;

            zR=tempZR;

            zI=tempZI;

            i++;

        }

        if (i==MAX\_ITERATION){

            return -1;

        }

        return i;

    }

}

3. Создание класса FractalExplorer, который позволит исследовать различные области фрактала, путем его создания, отображения через графический интерфейс Swing и обработки событий, вызванных взаимодействием приложения с пользователем.

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.event.MouseAdapter;

import java.awt.event.MouseEvent;

import java.awt.geom.Rectangle2D;

import javax.swing.JComponent;

import javax.swing.\*;

import java.awt.\*;

import java.awt.image.BufferedImage;

public class FractalExplorer {

    private int displaySize;

    private

    JImageDisplay display;

    private FractalGenerator fractal;

    private Rectangle2D.Double range;

    public FractalExplorer(int dSize) {

        displaySize = dSize;

        fractal = new Mandelbrot();

        range = new Rectangle2D.Double();

        fractal.getInitialRange(range);

        display = new JImageDisplay(displaySize, displaySize);

    }

    public void createAndShowGUI() {

        display.setLayout(new BorderLayout());

        JFrame frame = new JFrame("Фрактал");

        frame.add(display, BorderLayout.CENTER);

        JButton resetButton = new JButton("Очистить");

        ResetHandler handler = new ResetHandler();

        resetButton.addActionListener(handler);

        frame.add(resetButton, BorderLayout.SOUTH);

        MouseHandler click = new MouseHandler();

        display.addMouseListener(click);

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

        frame.pack();

        frame.setVisible(true);

        frame.setResizable(false);

    }

    private void drawFractal() {

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

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

                // методы статические

                double xCoord = FractalGenerator.getCoord(range.x,range.x + range.width, displaySize, x);

                double yCoord = FractalGenerator.getCoord(range.y,range.y + range.height, displaySize, y);

                int numIters = fractal.numIterations(xCoord, yCoord);

                if (numIters == -1){

                    display.drawPixel(x, y, 0);

                }

                else {

                    float hue = 0.7f + (float) numIters / 200f;

                    int rgbColor = Color.HSBtoRGB(hue, 1f, 1f);

                    display.drawPixel(x, y, rgbColor);

                }

            }

        } display.repaint();

    }

    private class ResetHandler implements ActionListener {

        public void actionPerformed(ActionEvent e) {

            fractal.getInitialRange(range);

            drawFractal();

        }

    }

    private class MouseHandler extends MouseAdapter {

        @Override

        public void mouseClicked(MouseEvent e) {

            int x = e.getX();

            double xCoord = FractalGenerator.getCoord(range.x,range.x + range.width, displaySize, x);

            int y = e.getY();

            double yCoord = FractalGenerator.getCoord(range.y,range.y + range.height, displaySize, y);

            fractal.recenterAndZoomRange(range, xCoord, yCoord, 0.5); drawFractal();

        }

    }

    public static void main(String[] args) {

        FractalExplorer displayExplorer = new FractalExplorer(800);

        displayExplorer.createAndShowGUI(); displayExplorer.drawFractal();

    }

}

Результат работы:

Если число итераций равно -1 (т.е. точка не выходит за границы, установите пиксель в черный цвет (для rgb значение 0). Иначе выберите значение цвета, основанное на количестве итераций. Можно также для этого использовать цветовое пространство HSV: поскольку значение цвета варьируется от 0 до 1, получается плавная последовательность цветов от красного к желтому, зеленому, синему, фиолетовому и затем обратно к красному! Для этого вы можете использовать следующий фрагмент: float hue = 0.7f + (float) numIters / 200f; int rgbColor = Color.HSBtoRGB(hue, 1f, 1f);



