**МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ**

**Ордена Трудового Красного Знамени**

**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Московский технический университет связи и информатики»**

Кафедра «Математическая кибернетика и информационные технологии»

**Отчет по лабораторной работе № 4**

по дисциплине «Информационные технологии и программирование»

на тему:

Рисование фракталов

Выполнила:

студентка группы БВТ2104

Степанова Дарина Викторовна

Проверила:

Мосева Марина Сергеевна

Москва, 2022

**Цель работы**: с помощью фреймворка Swing и Java API, который позволяет создавать графические пользовательские интерфейсы, создать небольшое JAVA-приложение, которое сможет рисовать фракталы.

**Задания:**

1. Создайте класс JImageDisplay, производный от javax.swing.JComponent. Класса должен иметь одно поле с типом доступа private, экземпляп java.awt.image.BufferedImage. Класс BufferedImage управляет изображением, содержимое которого можно записать.
2. Конструктор JImageDisplay должен принимать целочисленные значения ширины и высоты, и инициализировать объект BufferedImage новым изображением с этой шириной и высотой, и типом изображения TYPE\_INT\_RGB. Тип определяет, как цвета каждого пикселя будут представлены в изображении; значение TYPE\_INT\_RGB обозначает, что красные, зеленые и синие компоненты имеют по 8 битов, представленные в формате int в указанном порядке. Конструктор также должен вызвать метод setPreferredSize() родительского класса метод с указанной шириной и высотой. (Вы должны будете передать эти значения в объект java.awt.Dimension) Таким образом, когда ваш компонент будет включен в пользовательский интерфейс, он отобразит на экране все изображение.
3. Пользовательские компоненты Swing должны предоставлять свой собственный код для отрисовки, переопределяя защищенный метод JComponent paintComponent (Graphics g). Так как наш компонент просто выводит на экран данные изображения, реализация будет очень проста! Во-первых, нужно всегда вызывать метод суперкласса paintComponent (g) так, чтобы объекты отображались правильно. После вызова версии суперкласса, вы можете нарисовать изображение в компоненте, используя следующую операцию: g.drawImage (image, 0, 0, image.getWidth(), image.getHeight(), null); (Мы передаем значение null для ImageObserver, поскольку данная функциональность не требуется.)
4. Вы также должны создать два метода с доступом public для записи данных в изображение: метод clearImage (), который устанавливает все пиксели изображения в черный цвет (значение RGB 0), и метод drawPixel (int x, int y, int rgbColor), который устанавливает пиксель в определенный цвет. Оба метода будут необходимы для использования в методе setRGB () класса BufferedImage.
5. Для создания фракталов используйте следующий исходный файл FractalGenerator.java, от которого будут унаследованы все ваши фрактальные генераторы. Функция для фрактала Мандельброта имеет вид: zn = zn-1 2 + c, где все значения — это комплексные числа, z0 = 0, и с - определенная точка фрактала, которую мы отображаем на экране. Вычисления повторяются до тех пор, пока |z| > 2 (в данной ситуации точка находится не во множестве Мандельброта), или пока число итераций не достигнет максимального значения, например, 2000 (в этом случае делается предположение, что точка находится в наборе).
6. Создайте подкласс FractalGenerator с именем Mandelbrot. в нем вам необходимо будет обеспечить только два метода: getInitialRange() и numIterations().
7. getInitialRange (Rectangle2D.Double) - метод позволяет генератору фракталов определить наиболее «интересную» область комплексной плоскости для конкретного фрактала. Обратите внимание на то, что методу в качестве аргумента передается прямоугольный объект, и метод должен изменить поля прямоугольника для отображения правильного начального диапазона для фрактала. (Пример можно увидеть в методе FractalGenerator.recenterAndZoomRange().) В классе Mandelbrot этот метод должен установить начальный диапазон в (-2 - 1.5i) - (1 + 1.5i). Т.е. значения x и y будут равны -2 и -1.5 соответственно, а ширина и высота будут равны 3.
8. Метод numIterations(double, double) реализует итеративную функцию для фрактала Мандельброта. Константу с максимальным количеством итераций можно определить следующим образом:

public static final int MAX\_ITERATIONS = 2000

1. В случае, если алгоритм дошел до значения MAX\_ITERATIONS нужно вернуть -1, чтобы показать, что точка не выходит за границы.
2. 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. Все вышеприведенные поля будут иметь тип доступа private.

1. 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.
2. 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. Вам необходимо дать окну подходящий заголовок и обеспечить операцию закрытия окна по умолчанию (см. метод JFrame.setDefaultCloseOperation ()). После того, как компоненты пользовательского интерфейса инициализированы и размещены, добавьте следующую последовательность операций: frame.pack (); frame.setVisible (true); frame.setResizable (false); Данные операции правильно разметят содержимое окна, сделают его видимым (окна первоначально не отображаются при их создании для того, чтобы можно было сконфигурировать их прежде, чем выводить на экран), и затем запретят изменение размеров окна.
3. Реализуйте вспомогательный метод с типом доступа private для вывода на экран фрактала, можете дать ему имя drawFractal (). Этот метод должен циклически проходить через каждый пиксель в отображении (т.е. значения x и y будут меняться от 0 до размера отображения), и сделайте следующее: ¬ Вычислите количество итераций для соответствующих координат в области отображения фрактала. Вы можете определить координаты с плавающей точкой для определенного набора координат пикселей, используя вспомогательный метод FractalGenerator.getCoord (); например, чтобы получить координату x, соответствующую координате пикселя X, сделайте следующее: //x - пиксельная координата; xCoord - координата в пространстве фрактала double xCoord = FractalGenerator.getCoord (range.x, range.x + range.width, displaySize, x); ¬ 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); Если вы придумали другой способ отображения пикселей в зависимости от количества итераций, попробуйте реализовать его! ¬ Отображение необходимо обновлять в соответствии с цветом для каждого пикселя. ¬ После того, как вы закончили отрисовывать все пиксели, вам необходимо обновить JimageDisplay в соответствии с текущим изображением. Для этого вызовите функцию repaint() для компонента. В случае, если вы не воспользуетесь данным методом, изображение на экране не будет обновляться!
4. 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. После того, как вы создали этот класс, обновите метод createAndShowGUI ().
5. 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. При получении события о щелчке мышью, класс должен отобразить пиксельные кооринаты щелчка в область фрактала, а затем вызвать метод генератора recenterAndZoomRange() с координатами, по которым щелкнули, и масштабом 0.5. Таким образом, нажимая на какое-либо место на фрактальном отображении, вы увеличиваете его! Не забывайте перерисовывать фрактал после того, как вы меняете область фрактала. Далее обновите метод createAndShowGUI (), чтобы зарегистрировать экземпляр этого обработчика в компоненте фрактального отображения.
6. В заключении, вам необходимо создать статический метод main() для FractalExplorer так, чтобы можно было его запустить. В main необходимо будет сделать: ¬ Инициализировать новый экземпляр класса FractalExplorer с размером отображения 800. ¬ Вызовите метод createAndShowGUI () класса FractalExplorer. ¬ Вызовите метод drawFractal() класса FractalExplorer для отображения начального представления.

**Код программы:**

1. Файл FractalExplorer.java

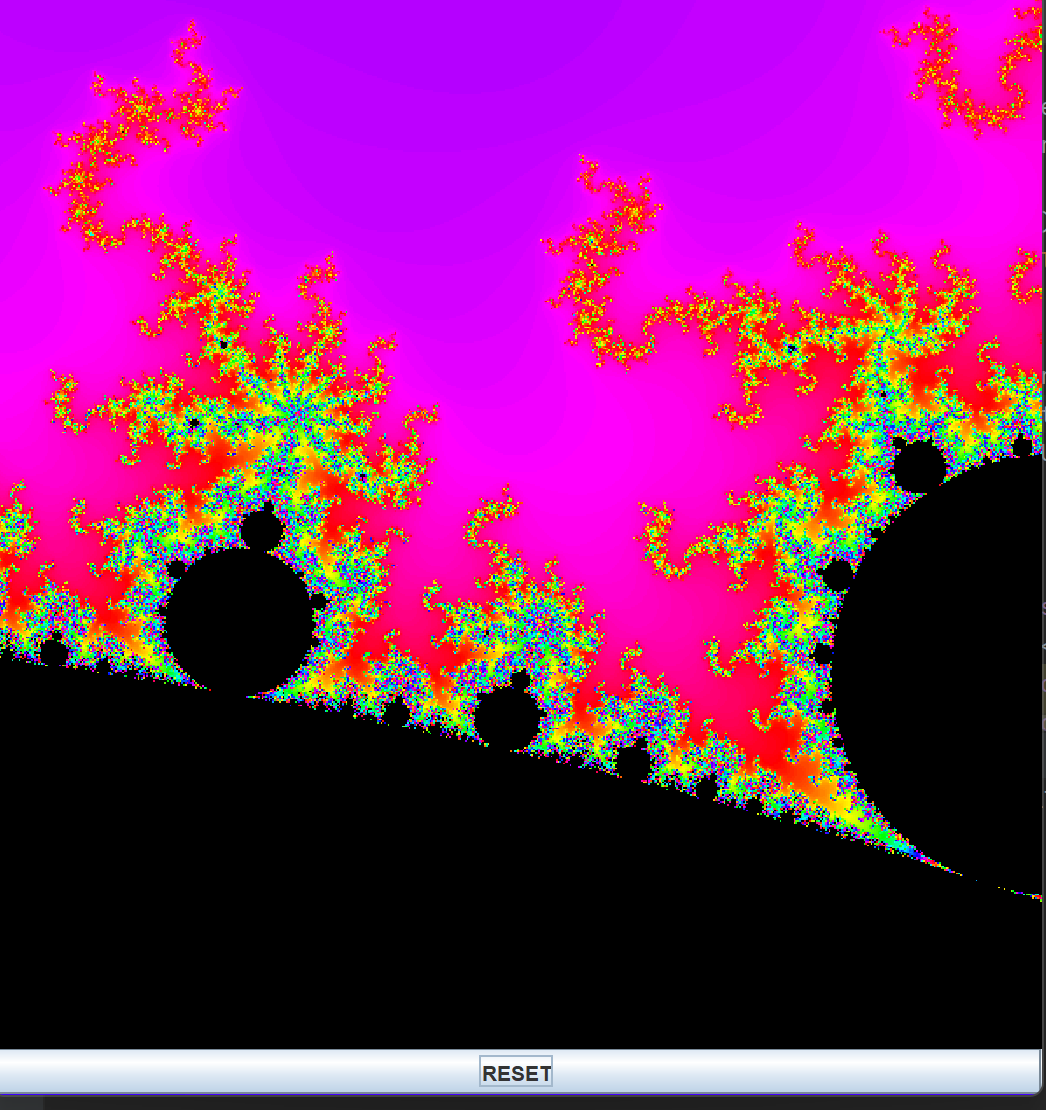
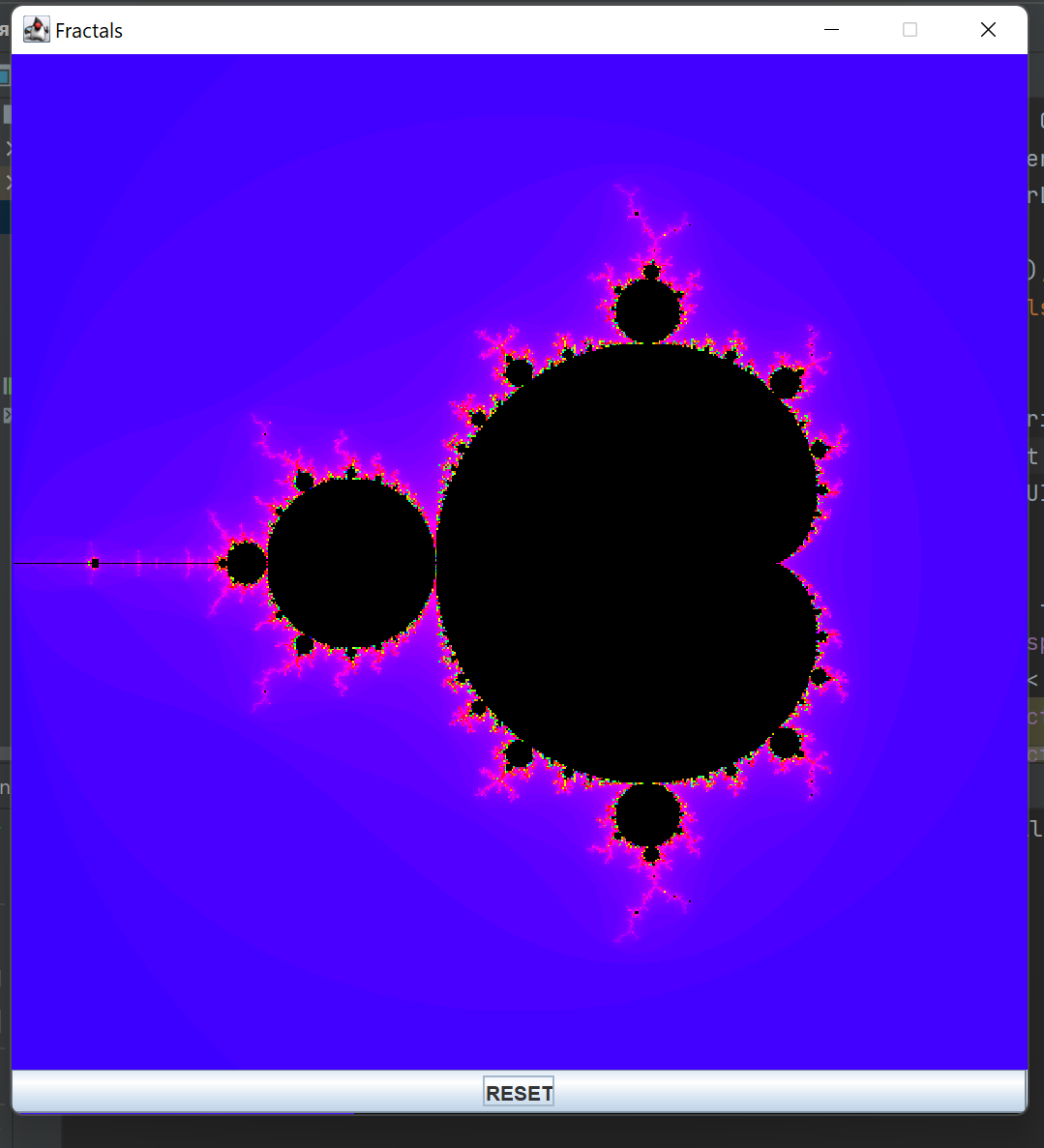
import java.awt.\*;  
import javax.swing.\*;  
import java.awt.geom.Rectangle2D;  
import java.awt.event.\*;  
public class FractalExplorer {  
 private int displaySize; //сторона квадрата в пикселях  
 private JImageDisplay display; //рисовать пиксели, стирать, работа с изображением  
 private FractalGenerator fractal; //  
 private Rectangle2D.Double initRange; //класс для работы с прямоугольникамиб переменная хранит координаты х и у, высоту и ширину  
 public FractalExplorer (int displaySize) {  
 this.displaySize = displaySize;  
 fractal = new Mandelbrot(); //т.к. мондельброт наследуется от фрактал.генератора, мы можем поместить объект класса мондельброт в переменную с типом данных фракт-ген  
 initRange = new Rectangle2D.Double();  
 fractal.getInitialRange(initRange);  
 }  
 class ResetButton implements ActionListener {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 fractal.getInitialRange(initRange);  
 drawFractal();  
 }  
 }  
 class ClickMouse extends MouseAdapter {  
 @Override  
 public void mouseClicked(MouseEvent event) {  
 display.clearImage();  
 int x = event.getX();  
 int y = event.getY();  
 double xF = fractal.*getCoord*(initRange.x, initRange.x + initRange.width, displaySize, x);  
 double yF = fractal.*getCoord*(initRange.y, initRange.y + initRange.height, displaySize, y);  
 fractal.recenterAndZoomRange(initRange, xF, yF, 0.5);  
 drawFractal();  
 }  
 }  
 public void createAndShowGUI() {  
 JFrame frame = new JFrame("Fractals");  
 frame.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 display = new JImageDisplay(displaySize, displaySize); //создается объект для работы с изображением  
 frame.add(display, BorderLayout.*CENTER*); //прикрепляем картинку к окну приложения по центру  
 JButton reset = new JButton("RESET");  
 ResetButton resetBut = new ResetButton();  
 reset.addActionListener(resetBut);  
 ClickMouse click = new ClickMouse();  
 display.addMouseListener(click);  
 frame.add(reset, BorderLayout.*SOUTH*);  
 frame.pack ();  
 frame.setVisible (true);  
 frame.setResizable (false);  
 drawFractal();  
 }  
 public static void main(String[] args) {  
 FractalExplorer goPaint = new FractalExplorer(600);  
 goPaint.createAndShowGUI();  
 }  
 private void drawFractal() {  
 for (int i = 0; i < displaySize; i++) {  
 for (int j = 0; j < displaySize; j++) {  
 double x = fractal.*getCoord*(initRange.x, initRange.x + initRange.width, displaySize, i);  
 double y = fractal.*getCoord*(initRange.y, initRange.y + initRange.height, displaySize, j);  
 int iteration = fractal.numIterations(x, y); //сколько итераций у фрактала в данной точке  
 if (iteration == -1) {  
 display.drawPixel(i, j, 0);  
 }  
 else {  
 float hue = 0.7f + (float) iteration / 200f;  
 int rgbColor = Color.*HSBtoRGB*(hue, 1f, 1f); //чем больше итераций, тем светлее цвет  
 display.drawPixel(i, j, rgbColor);  
 }  
 }  
 }  
 display.repaint();  
 }  
}

1. Файл Mandelbrot.java

import java.awt.geom.Rectangle2D;  
public class Mandelbrot extends FractalGenerator{  
 public static final int *MAX\_ITERATIONS* = 2000;  
 @Override  
 public void getInitialRange(Rectangle2D.Double initRange) { //указываем начальные координаты от которых будет отрисовываться фрактал, попадет в обзор  
 initRange.x = -2;  
 initRange.y = -1.5;  
 initRange.width = 3;  
 initRange.height = 3;  
 }  
 @Override  
 public int numIterations(double x, double y) {//считает кол-во итераций для фрактала, сколько можно  
 int countOfIter = 0;  
 double zRe = 0;  
 double zIm = 0;  
 double zRe2 = 0;  
 double zIm2 = 0;  
 while ((countOfIter < *MAX\_ITERATIONS*) && ((zRe2 + zIm2) < 4)) {  
 zIm = (2 \* zRe \* zIm) + y;  
 zRe = (zRe2 - zIm2) + x;  
 zIm2 = zIm \* zIm;  
 zRe2 = zRe \* zRe;  
 countOfIter++;  
 }  
 if (countOfIter == *MAX\_ITERATIONS*) return -1;  
 else return countOfIter;  
 }  
}

1. Файл JImageDisplay.java
2. import java.awt.image.BufferedImage;  
   import java.awt.\*;  
   public class JImageDisplay extends javax.swing.JComponent{  
    private BufferedImage bufIm;  
    public JImageDisplay(int width, int height) {  
    bufIm = new BufferedImage(width, height, BufferedImage.*TYPE\_INT\_RGB*);//создаем объект класса баф  
    Dimension dim = new Dimension(width, height); //разрешение  
    super.setPreferredSize(dim);//загружаем в род. класс разрешение  
    }  
    @Override  
    public void paintComponent(Graphics g) {  
    super.paintComponent(g);  
    g.drawImage(bufIm, 0, 0, bufIm.getWidth(), bufIm.getHeight(), null);  
    }  
    public void clearImage() {  
    int[] colourBlack = new int[bufIm.getWidth()\*bufIm.getHeight()];  
    bufIm.setRGB(0, 0, bufIm.getWidth(), bufIm.getWidth(),colourBlack, 0, 1); //полность закрашивает экран в черный, т.к массив пустой  
    }  
    public void drawPixel(int x, int y, int colour) {  
    bufIm.setRGB(x, y, colour);  
    }  
   }

**Скриншоты выполнения:**



**Ссылка на репозиторий GitHub:** https://github.com/DarinaStepanova/iteh

**Заключение:**

Таким образом, я с помощью фреймворка Swing и Java API, который позволяет создавать графические пользовательские интерфейсы, создала небольшое JAVA-приложение, которое может рисовать фракталы.