Федеральное агенство связи

Ордена Трудового Красного Знамени федеральное государственное

бюджетное учреждение высшего образования

«Московский технический университет связи и информатики»

Кафедра Математической кибернетики и

информационных технологий

Лабораторная работа №4

по дисциплине: «Рисование фракталов»

Выполнил студент

группы БФИ1902

Рахимов Е.К.

Проверила:

Мосева М.С.

Москва, 2020 г.

Оглавление

[1. Цель лабораторной работы 2](#_Toc58524139)

[2. Задание на лабораторную работу 3](#_Toc58524140)

[3. Ход лабораторной работы 4](#_Toc58524141)

[3.1 Листинг программы 4](#_Toc58524142)

[3.2 Результат выполнения программы 7](#_Toc58524143)

[Список использованных источников 8](#_Toc58524144)

# 1. Цель лабораторной работы

Цель данной лабораторной работы — научиться создавать интерфейс и работать с интерфейсом , а также рисовать фракталы.

# 2. Задание на лабораторную работу

1) **JImageDisplay**:

• 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.

• 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.

• 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.)

• 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.

**2) Mandelbrot:**

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

• 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.

• 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; Затем вы сможете ссылаться на эту переменную в вашей реализации. Обратите внимание на то, что у Java нет подходящего типа данных для комплексных чисел, поэтому необходимо будет реализовать итеративную функцию, используя отдельные переменные для действительной и мнимой частей. (Вы можете реализовать отдельный класс для комплексных чисел.) Ваш алгоритм должен обладать быстродействием, например, не стоит сравнивать |z| с 2; сравните |z|2 с 22 для того, чтобы избежать сложных и медленных вычислений квадратного корня. Также не стоит использовать метод Math.pow () для вычисления небольших степеней, лучше перемножьте значение, иначе ваш быстродействие вашего кода сильно упадет. В случае, если алгоритм дошел до значения MAX\_ITERATIONS нужно вернуть -1, чтобы показать, что точка не выходит за границы

**3)** **FractalExplorer:**

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.

• 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 какиелибо 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. 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.

• 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!

• 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 ().

• 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.

• 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

# 3. Ход лабораторной работы

## 3.1 Листинг программы

package lab4;  
  
import java.awt.BorderLayout;  
import java.awt.Color;  
import java.awt.Container;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.awt.event.MouseEvent;  
import java.awt.event.MouseAdapter;  
import java.awt.geom.Rectangle2D;  
import java.io.File;  
import java.io.IOException;  
  
import javax.imageio.ImageIO;  
import javax.swing.JButton;  
import javax.swing.JComboBox;  
import javax.swing.JFileChooser;  
import javax.swing.JFrame;  
import javax.swing.JLabel;  
import javax.swing.JOptionPane;  
import javax.swing.JPanel;  
import javax.swing.filechooser.FileNameExtensionFilter;  
  
import lab4.fractal.\*;  
  
public class FractalExplorer {  
   
 */\*\*  
 размер экрана  
 \*/* private int dispSize;  
   
 */\*\*  
 Ссылка для обновления отображения в разных методах в процессе вычисления фрактала  
 \*/* private JImageDisplay img;  
   
  
 private JComboBox<String> fractalChos;  
   
  
 private JFrame frame;  
   
 */\*\*  
 Ссылка на базовый класс для отображения других видов фракталов в будущем  
 \*/* private FractalGenerator generetion;  
   
 */\*\*  
 Объект Rectangle2D.Double, указывающий диапазона комплексной  
 плоскости, которая выводится на экран.  
 \*/* Rectangle2D.Double rng;  
  
 private class FractalHandler implements ActionListener   
 {   
 public void actionPerformed(ActionEvent e)   
 {   
 String cmd = e.getActionCommand();   
  
 if (e.getSource() == fractalChos)  
 {   
 String selectedItem = fractalChos.getSelectedItem().toString();  
  
 if(selectedItem.equals(Mandelbrot.*nameString*()))  
 {  
 generetion = new Mandelbrot();  
 }  
  
 else  
 {  
 JOptionPane.*showMessageDialog*(null, "Error: fractalChooser unknown choice");  
 return;  
 }  
   
 rng = new Rectangle2D.Double();  
 generetion.getInitialRange(rng);  
   
 drawFractal();  
 }   
 else if (cmd.equals("reset"))   
 {   
 rng = new Rectangle2D.Double();  
 generetion.getInitialRange(rng);  
   
 drawFractal();  
 }   
 else if (cmd.equals("save"))   
 {   
 JFileChooser chooser = new JFileChooser();  
   
 FileNameExtensionFilter filter = new FileNameExtensionFilter("PNG Images", "png");  
 chooser.setFileFilter(filter);  
 chooser.setAcceptAllFileFilterUsed(false);  
   
 if(chooser.showSaveDialog(null) == JFileChooser.*APPROVE\_OPTION*)  
 {  
 try   
 {  
 File fd = chooser.getSelectedFile();  
 String filePath = fd.getPath();  
 if(!filePath.toLowerCase().endsWith(".png"))  
 {  
 fd = new File(filePath + ".png");  
 }  
   
 ImageIO.*write*(img.getImage(), "png", fd);  
 }   
 catch (IOException exc)   
 {  
 JOptionPane.*showMessageDialog*(null, "Error: couldn't save file ( " + exc.getMessage() + " )");  
   
 exc.printStackTrace();  
 }  
 }  
 }   
 else  
 {  
 JOptionPane.*showMessageDialog*(null, "Error: FractalHandler unknown action");  
 }  
 }   
 }  
 // для обрабатывания работы мыши - увеличение  
 private class MouseHandler extends MouseAdapter   
 {   
 public void mouseClicked(MouseEvent e)  
 {  
 double xCoord = getFractlXcord(e.getX());  
 double yCoord = getFractlYcord(e.getY());  
   
 generetion.recenterAndZoomRange(rng,xCoord, yCoord, 0.5);  
   
 drawFractal();  
 }   
 }  
 */\*\*  
 конструктор, который принимает значение  
 размера отображения в качестве аргумента, затем сохраняет это значение в  
 соответствующем поле, а также инициализирует объекты диапазона и  
 фрактального генератора.  
 \*/* public FractalExplorer(int displaySize)  
 {  
 dispSize = displaySize;  
   
 generetion = new Mandelbrot();  
   
 rng = new Rectangle2D.Double();  
 generetion.getInitialRange(rng);  
 }  
 */\*\*  
 Метод createAndShowGUI () инициализирует  
 графический интерфейс Swing: JFrame, содержащий объект JimageDisplay, и  
 кнопку для сброса отображения.  
 \*/* public void createAndShowGUI()  
 {  
 frame = new JFrame("Fractal Explorer");//дать окну подходящий заголовок и  
  
 frame.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);//обеспечить операцию закрытия окна по умолчанию  
 Container contentpn = frame.getContentPane();//Создаю панель содержимого  
  
 contentpn.setLayout(new BorderLayout()); // устанавливаю расположение границ  
   
 FractalHandler handler = new FractalHandler();// создаю объект класса fractalhandler  
   
 */\*\*  
 \* Выбираю фрактальную панель  
 \*/* JPanel fractalPanel = new JPanel();// создаю фрактальную панель  
   
 JLabel panelLabel = new JLabel("Fractal: ");// создаю название фрактальной панели  
 fractalPanel.add(panelLabel);// добавляю название к фрактальной панеле  
 //Беру названия  
 fractalChos = new JComboBox<String>();  
 fractalChos.addItem(Mandelbrot.*nameString*());  
 fractalChos.addActionListener(handler);  
   
 fractalPanel.add(fractalChos);// добавляю названия в фрактальную панель  
   
 contentpn.add(fractalPanel, BorderLayout.*NORTH*);//Добавляю к панеле содержимого фрактальную панель с расположением BorderLayout.NORTH  
  
 // создаю изображение и добавляю на позицию BorderLayout.CENTER  
 img = new JImageDisplay(dispSize, dispSize);  
 contentpn.add(img, BorderLayout.*CENTER*);  
   
 //Создаю кнопку для панели  
 JPanel buttonsPanel = new JPanel();  
   
 //Создаю кнопку для сохранения изображения  
 JButton saveButton = new JButton("Save Image");  
 saveButton.setActionCommand("save");   
 saveButton.addActionListener(handler);  
 buttonsPanel.add(saveButton);  
   
 //Создаю кнопку для сброса дисплея  
 JButton resetButton = new JButton("Reset Display");  
 resetButton.setActionCommand("reset");   
 resetButton.addActionListener(handler);  
 buttonsPanel.add(resetButton);  
   
 contentpn.add(buttonsPanel, BorderLayout.*SOUTH*);//Добавляю к панели содержимого панель кнопок на позицию BorderLayout.SOUTH  
   
 contentpn.addMouseListener(new MouseHandler());//добавляю MouseHandler для того, чтобы работала компьютерная мыш и  
 // происходило отслеживание действий  
  
  
 //Данные операции правильно разметят содержимое окна, сделают его  
 //видимым и затем запретят изменение размеров окна.  
 frame.pack();  
 frame.setVisible(true);  
 frame.setResizable(false);  
 }  
 //вывода на экран фрактала. данный метод циклически проходит через каждый пиксель в отображении  
 public void drawFractal()  
 {  
 double xcord = 0;  
 double ycord = 0;  
   
 float numiter = 0;  
 float hue = 0;  
   
 int rgbColor = 0;  
 // циклическая обработка пикселей  
 for(int x = 0; x < dispSize; ++x)  
 {  
 xcord = getFractlXcord(x);  
   
 for(int y = 0; y < dispSize; ++y)  
 {  
 ycord = getFractlYcord(y);  
   
 numiter = generetion.numIterations(xcord, ycord);  
 if(numiter < 0)  
 {  
 rgbColor = 0;  
 }  
 else  
 {  
 hue = 0.7f + numiter / 200f;  
 rgbColor = Color.*HSBtoRGB*(hue, 1f, 1f);  
 }  
 //отображаю пиксели  
 img.drawPixel(x, y, rgbColor);  
 }  
 }  
  
 img.repaint();// обновляю изображение  
 }  
   
 */\*\*  
 Метод который возвращает координату в пространстве фрактала для х  
 \*/* private double getFractlXcord(int x)  
 {  
 return FractalGenerator.*getCoord*(rng.x, rng.x + rng.width, dispSize, x);  
 }  
  
 */\*\*  
 Метод который возвращает координату в пространстве фрактала для у  
 \*/* private double getFractlYcord(int y)  
 {  
 return FractalGenerator.*getCoord*(rng.y, rng.y + rng.height, dispSize, y);  
 }  
   
 */\*\*  
 \* Entry-point for the application. No command-line arguments are  
 \* recognized at this time.  
 \*\*/* public static void main(String[] args)   
 {  
 FractalExplorer explorer = new FractalExplorer (400);//Инициализировую новый экземпляр класса FractalExplorer с  
 //размером отображения 800  
 explorer.createAndShowGUI();//отображаю интерфейс  
 explorer.drawFractal();// рисую фрактал  
 }   
}

package lab4;  
  
import java.awt.geom.Rectangle2D;  
  
  
//Этот класс предоставляет общий интерфейс и операции для фрактальных  
//генераторов, которые можно просмотреть в FractalExplorer  
public abstract class FractalGenerator {  
  
 */\*\*  
 Эта статическая вспомогательная функция принимает целочисленную координату и преобразует ее  
 в значение double, соответствующее определенному диапазону. Он  
 используется для преобразования координат пикселей в значения двойной точности для  
 вычисление фракталов и т. д.  
 range Min - минимальное значение диапазона с плавающей запятой  
 rangeMax - максимальное значение диапазона с плавающей запятой  
 size - размер измерения, из которого исходит координата пикселя.  
 Например, это может быть ширина изображения или высота изображения.  
 coord - координата для вычисления значения двойной точности.  
 Координата должна находиться в диапазоне [0, размер].  
 \*/* public static double getCoord(double rngMin, double rngMax,  
 int size, int coord) {  
  
 assert size > 0;  
 assert coord >= 0 && coord < size;  
  
 double range = rngMax - rngMin;  
 return rngMin + (range \* (double) coord / (double) size);  
 }  
  
  
 */\*\*Метод позволяет генератору фракталов определить наиболее «интересную» область комплексной плоскости  
 для конкретного фрактала. Задает заданный прямоугольник, содержащий начальный диапазон, подходящий для генерируемого фрактала.  
 \*/* public abstract void getInitialRange(Rectangle2D.Double range);  
  
  
 */\*\*  
 Обновляет текущий диапазон для центрирования по заданным координатам  
 и увеличения или уменьшения масштаба с заданным коэффициентом масштабирования.  
 \*/* 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;  
 }  
  
  
 */\*\*  
 Учитывая координату <em>x</em> + <em>iy</em> в комплексной плоскости,  
 вычисляет и возвращает число итераций перед фракталом  
 функция экранирует ограничивающую область для этой точки. Указывается точка, которая  
 не убегает до достижения предела итерации  
 с результатом -1.  
 \*/* public abstract int numIterations(double x, double y);  
}

package lab4;  
  
import javax.swing.\*;  
import java.awt.\*;  
  
import java.awt.image.BufferedImage;  
  
public class JImageDisplay extends JComponent{//класс JImageDisplay, производный от javax.swing.JComponent//  
  
 private BufferedImage img;//класс BufferedImage управляет изображением, содержимое которого можно записать  
   
 public JImageDisplay(int width, int height)  
 {  
 img = new BufferedImage(width, height, BufferedImage.*TYPE\_INT\_RGB*);//инициализирую объект класса BufferedImage с данными характеристиками  
 //длинны и высоты, а также с типом изображения 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  
   
 Dimension dim = new Dimension(width, height);// создаю объект класса Dimension с данными характеристиками,чтобы в дальнейшем передать  
 // родительскому классу  
 super.setPreferredSize(dim);  
 }  
 //Пользовательские компоненты Swing должны предоставлять свой  
 // собственный код для отрисовки, переопределяя защищенный метод JComponent  
 //paintComponent (Graphics g)  
 protected void paintComponent(Graphics g)  
  
 {  
 super.paintComponent(g);//нужно всегда  
 //вызывать метод суперкласса paintComponent (g) так, чтобы объекты  
 //отображались правильно.  
  
 g.drawImage(img, 0, 0, img.getWidth(), img.getHeight(), null);//Рисую изображение  
 }  
   
  
 public void clearImage() //метод устанавливает все пиксели изображения в черный цвет  
 {  
 for(int j = 0 ; j < img.getHeight() ; ++j)  
 {  
 for(int i = 0 ; i < img.getWidth() ; ++i)  
 {  
 this.drawPixel(i, j, 0);  
 }  
 }  
 }  
 //метод устанавливает пиксель определенного цвета  
 public void drawPixel(int x, int y, int color)  
 {  
 img.setRGB(x, y, color);  
 }  
 //метод для получения картинки  
 public BufferedImage getImage()  
 {  
 return img;  
 }  
   
}

package lab4.fractal;  
  
import java.awt.geom.Rectangle2D;  
  
import lab4.FractalGenerator;  
  
public class Mandelbrot extends FractalGenerator{  
  
 public static final int *MAX\_ITERATIONS* = 2000;  
   
 */\*\*  
 \* установите начальный диапазон в (-2-1.5 i) - (1 + 1.5 i).  
 \* То есть значения x и y будут равны -2 и -1,5 соответственно,  
 \* а ширина и высота будут равны 3  
 \*/* public void getInitialRange(Rectangle2D.Double range)  
 {  
 range.x = -2;  
 range.y = -1.5;  
   
 range.width = 3;  
 range.height = 3;  
 }  
   
 */\*\*  
 \* Для фрактала Мандельброта функция z\_n = (z\_ (n-1))^2 + c,  
 \* где все значения являются комплексными числами, z\_0 = 0,  
 \* и c-конкретная точка фрактала, которую мы показываем.  
 \* Это вычисление повторяется до тех пор, пока либо |z| > 2  
 \* или до тех пор, пока число итераций не достигнет максимального значения, например 2000  
 \*/* public int numIterations(double x, double y)  
 {  
 int count = 0;  
   
 double re = x;  
 double im = y;  
   
 double re2 = re\*re;  
 double im2 = im\*im;  
   
 double z\_n2 = 0;  
   
 while(count < *MAX\_ITERATIONS* && z\_n2 < 4)  
 {  
 im = (2 \* re \* im) + y;  
 re = (re2 - im2) + x;  
   
 re2 = re\*re;  
 im2 = im\*im;  
   
 z\_n2 = re2 + im2;  
 ++count;  
 }  
   
 return count < *MAX\_ITERATIONS* ? count : -1;  
 }  
   
 public static String nameString()  
 {  
 return "Mandelbrot";  
 }  
   
}

## 3.2 Результат выполнения программы

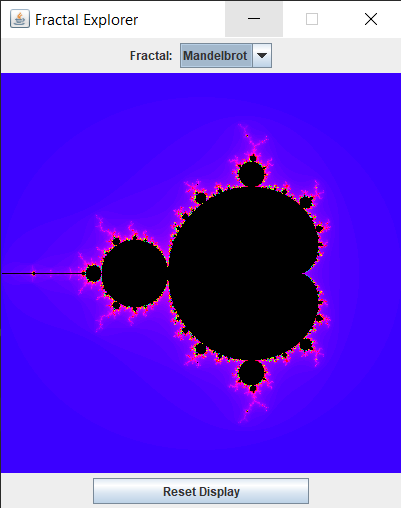


Рисунок 1 – результат выполнения

# Список использованных источников

1) ГОСТ 7.32-2017 Система стандартов по информации, библиотечному и издательскому делу. Отчёт о научно-исследовательской работе. Структура и правила оформления

2) ГОСТ 7.1-2003 Библиографическая запись. Библиографическое описание. Общие требования и правила составления