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 №5

по дисциплине: «Выбор и сохранение 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) **FractalGenerator**:

• Создать 2 новые реализации FractalGenerator Первым будет фрактал tricorn, который должен находиться в файле Tricorn.java. Для этого нужно создать подкласс FractalGenerator и реализация будет почти идентична фракталу Мандельброта, кроме двух изменений. Вы даже можете скопировать исходный код фрактала Мандельберта и просто внести следующие изменения: ¬ Уравнение имеет вид zn = zn-1 2 + c. Единственное отличие только в том, что используется комплексное сопряжение zn-1 на каждой итерации. ¬ Начальный диапазон для трехцветного фрактала должен быть от (- 2, -2) до (2, 2). Второй фрактал, который необходимо реализовать - это фрактал «Burning Ship», который в реальности не похож на пылающий корабль. Данный фрактал имеет следующие свойства: ¬ Уравнение имеет вид zn = (|Re(zn-1)| + i |Im(zn-1)|)2 + c. Другими словами, вы берете абсолютное значение каждого компонента zn-1 на каждой итерации. ¬ Начальный диапазон для данного фрактала должен быть от (-2, -2.5) до (2, 1.5).

• Сombo-boxe в Swing может управлять коллекцией объектов, но объекты должны предоставлять метод toString(). Убедитесь, что в каждой реализации фракталов tcnm метод toString(), который возвращает имя, например «Mandelbrot», «Tricorn» и «Burning Ship».

• Настроить JComboBox в вашем пользовательском интерфейсе можно с использованием конструктора без параметров, а затем использовать метод addItem(Object) для того, чтобы добавить реализации вашего генератора фракталов. Как указывалось в предыдущем шаге, выпадающий список будет использовать метод toString () в ваших реализациях для отображения генераторов в выпадающем списке. Необходимо будет также добавить объект label в разрабатываемый пользовательский интерфейс перед выпадающим списком, в качестве пояснения к выпадающему списку. Это можно сделать, создав новый объект Jpanel и добавив в него объекты JLabel и JcomboBox, а затем разместить панель на позиции NORTH на вашем макете окна. И наконец, необходимо добавить поддержку выпадающего списка в реализацию ActionListener. В случае, если событие поступило от выпадающего списка, вы можете извлечь выбранный элемент из виджета и установить его в качестве текущего генератора фракталов. (Используйте метод getSelectedItem())

**2) Сохранения изображения Фрактала:**

Следующая ваша задача - сохранение текущего изображения фрактала на диск. Java API предоставляет несколько инструментов для реализации данной задачи.

• Во-первых, вам нужно добавить кнопку «Save Image» в ваше окно. Для этого вы можете добавить обе кнопки «Save Image» и «Reset» в новую Jpanel, а затем разместить эту панель в SOUTH части окна. События от кнопки «Save Image» также должны обрабатываться реализацией ActionListener. Назначьте кнопкам «Save Image» и «Reset» свои значения команд (например, «save» и «reset») для того, чтобы обработчик событий мог отличить события от этих двух разных кнопок.

• В обработчике кнопки «Save Image» вам необходимо реализовать возможность указания пользователем, в какой файл он будет сохранять изображение. Это можно сделать с помощью класса javax.swing.JFileChooser. Указанный класс предоставляет метод showSaveDialog(), который открывает диалоговое окно «Save file», позволяя тем самым пользователю выбрать директорию для сохранения. Метод принимает графический компонент, который является родительским элементом для диалогового окна с выбором файла, что позволяет центрированию окна с выбором относительно его родителя. В качестве родителя используйте окно приложения. Как вы могли заметить, данный метод возвращает значение типа int, которое указывает результат операции выбора файла. Если метод возвращает значение JfileChooser.APPROVE\_OPTION, тогда можно продолжить операцию сохранения файлов, в противном случае, пользователь отменил операцию, поэтому закончите данную обработку события без сохранения. Если пользователь выбрал директорию для сохранения файла, вы можете ее узнать, используя метод getSelectedFile(), который возвращает объект типа File.

• Также необходимо настроить средство выбора файлов, чтобы сохранять изображения только в формате PNG, на данном этапе вы будете работать только с данным форматом. вы сможете это настроить с помощью javax.swing.filechooser.FileNameExtensionFilter, как это продемонстрировано ниже: JFileChooser chooser = new JfileChooser(); FileFilter filter = new FileNameExtensionFilter("PNG Images", "png"); chooser.setFileFilter(filter); chooser.setAcceptAllFileFilterUsed(false); Последняя строка гарантирует, что средство выбора не разрешит пользователю использование отличных от png форматов.

• Если пользователь успешно выбрал файл, следующим шагом является сохранения изображения фрактала на диск! Для данного рода задач Java включает в себя необходимую функциональность. Класс javax.imageio.ImageIO обеспечивает простые операции загрузки и сохранения изображения. Вы можете использовать метод write(RenderedImage im, String formatName, File output). Параметр formatName будет содержать значение «png”. Тип «RenderedImage» - это просто экземпляр BufferedImage из вашего компонента JimageDisplay. (Используйте для него тип доступа public) Метод write() может вызвать исключение, поэтому вам необходимо заключить этот вызов в блок try/catch и обработать возможную ошибку. Блок catch должен проинформировать пользователя об ошибке через диалоговое окно. Swing предоставляет класс javax.swing.JoptionPane для того, чтобы упростить процесс создания информационных диалоговых окон или окон, где нужно выбрать да/нет. Для этого вы можете использовать статический метод JoptionPane.showMessageDialog(Component parent, Object message, String title, int messageType), где messageType у вас будет JOptionPane.ERROR\_MESSAGE. В сообщении об ошибке вы можете использовать возвращаемое значение метода getMessage(), а заголовком окна может быть, например, «Cannot Save Image». Родительским компонентом будет окно для того, чтобы диалоговое окно с сообщением об ошибке выводилось относительно центра окна

# 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 if(selectedItem.equals(Tricorn.*nameString*()))  
 {  
 generetion = new Tricorn();  
 }  
 else if(selectedItem.equals(BurningShip.*nameString*()))  
 {  
 generetion = new BurningShip();  
 }  
 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.addItem(BurningShip.*nameString*());  
 fractalChos.addItem(Tricorn.*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 BurningShip extends FractalGenerator{  
  
 public static final int MAX\_ITERATIONS = 2000;  
   
 */\*\*  
 \* Начальный диапазон для этого фрактала должен быть от (-2, -2,5) до (2, 1,5)  
 \* То есть значения x и y будут равны -2 и -2,5 соответственно,  
 \* а ширина и высота будут равны 4  
 \*/* public void getInitialRange(Rectangle2D.Double range)  
 {  
 range.x = -2;  
 range.y = -2.5;  
   
 range.width = 4;  
 range.height = 4;  
 }  
   
 */\*\*  
 \* Для фрактала Мандельброта, функция z\_n = (|Re(z\_(n-1))| + i\* |Im(z\_(n-1))|)^2 + c,  
 \* \* где все величины являются комплексными числами, z\_0 = 0,  
 \* \* и C-это конкретная точка в фрактал, который 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 = Math.abs(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 "Burning Ship";  
 }  
   
}

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";  
 }  
   
}

package lab4.fractal;  
  
import java.awt.geom.Rectangle2D;  
  
import lab4.FractalGenerator;  
  
public class BurningShip extends FractalGenerator{  
  
 public static final int *MAX\_ITERATIONS* = 2000;  
   
 */\*\*  
 \* Начальный диапазон для этого фрактала должен быть от (-2, -2,5) до (2, 1,5)  
 \* То есть значения x и y будут равны -2 и -2,5 соответственно,  
 \* а ширина и высота будут равны 4  
 \*/* public void getInitialRange(Rectangle2D.Double range)  
 {  
 range.x = -2;  
 range.y = -2.5;  
   
 range.width = 4;  
 range.height = 4;  
 }  
   
 */\*\*  
 \* Для фрактала Мандельброта, функция z\_n = (|Re(z\_(n-1))| + i\* |Im(z\_(n-1))|)^2 + c,  
 \* \* где все величины являются комплексными числами, z\_0 = 0,  
 \* \* и C-это конкретная точка в фрактал, который 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 = Math.*abs*(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 "Burning Ship";  
 }  
   
}

## 3.2 Результат выполнения программы

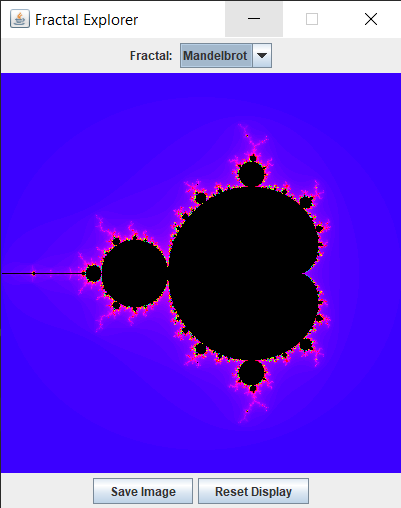


Рисунок 1 – результат выполнения

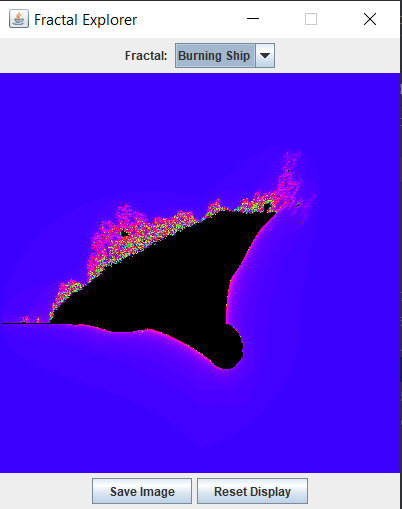


Рисунок 2 – результат выполнения

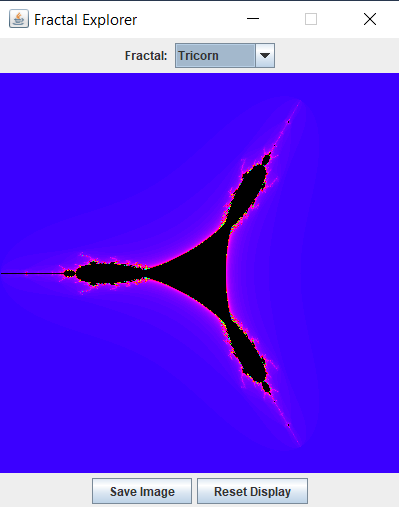


Рисунок 3 – результат выполнения

# 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