**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**

Факультет РиТ

**ЛАБОРАТОРНАЯ РАБОТА №5**

по дисциплине «Java-программирование»

на тему:

«Выбор и сохранение фракталов»

Выполнил: студ. гр. БПЗ1901

Неживлева Ксения

Проверил: Харрасов Камиль Раисович

Москва 2021

Цель работы:

В данной лабораторной работе генератор фракталов будет расширен двумя новыми функциями. Во-первых, вы добавите поддержку нескольких фракталов и реализуете возможность выбирать нужный фрактал из выпадающего списка. Во-вторых, вы добавите поддержку сохранения текущего изображения в файл.

Выполнение:

Так как в реализацию была введена абстракция FractalGenerator, добавление нескольких фракталов не будет проблемой. В данной лабораторной работе вы добавите поддержку нескольких фракталов, и пользователь сможет выбирать между ними, используя *combo-box*. Программный интерфейс Swing (Swing API) предоставляет *combo-box* через класс javax.swing.JComboBox, а также запускает ActionEvents при выборе новогоэлемента. Необходимо сделать:

* Создать 2 новые реализации FractalGenerator
* package com.lab5;  
  import java.awt.geom.Rectangle2D;  
    
  // Этот класс предоставляет общий интерфейс и операции для генераторов фракталов, которые можно просмотреть в Fractal Explorer.  
  public abstract class FractalGenerator  
  {  
   // Эта статическая вспомогательная функция принимает целочисленную координату и преобразует ее в значение двойной точности,  
   // соответствующее определенному диапазону. Он используется для преобразования координат пикселей в значения с двойной точностью  
   // для вычисления фракталов и т.д.  
   // @param rangeMin минимальное значение диапазона с плавающей запятой  
   // @param rangeMax максимальное значение диапазона с плавающей запятой  
   // @param size размер измерения, от которого происходит координата пикселя. Например, это может быть ширина изображения или высота изображения.  
   // @param - координата, для которой вычисляется значение двойной точности. Координата должна находиться в диапазоне [0, размер].  
   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);  
   }  
   // Устанавливает указанный прямоугольник, чтобы он содержал начальный диапазон, подходящий для генерируемого фрактала.  
   public abstract void getInitialRange(Rectangle2D.Double range);  
   // Обновляет текущий диапазон для центрирования по указанным координатам и увеличения или уменьшения масштаба  
   // с использованием указанного коэффициента 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 x + iy в комплексной плоскости, вычисляет и возвращает количество итераций до того,  
   // как фрактальная функция выйдет за ограничивающую область для этой точки.  
   // Точка, которая не исчезает до достижения предела итераций, обозначается результатом -1.  
   public abstract int numIterations(double x, double y);

Рис.1-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*-12 + *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).*

package com.lab5;  
import java.awt.geom.Rectangle2D;  
  
// Этот класс является подклассом FractalGenerator. Он используется для вычисления фрактала tricorn.  
public class Tricorn extends FractalGenerator  
{  
 // Константа для количества максимальных итераций.  
 public static final int MAX\_ITERATIONS = 2000;  
 */\*\* Этот метод позволяет генератору фракталов указать, какая часть комплексной плоскости наиболее интересна для фрактала.  
 \* Ему передается объект прямоугольника, и метод изменяет поля прямоугольника,  
 \* чтобы показать правильный начальный диапазон для фрактала.  
 \* Эта реализация устанавливает начальный диапазон в (-2 - 2i), (2 + 2i) или x = -2, y = -2, width = height = 4. \*/* public void getInitialRange(Rectangle2D.Double range)  
 {  
 range.x = -2;  
 range.y = -2;  
 range.width = 4;  
 range.height = 4;  
 }  
 */\*\* Этот метод реализует итерационную функцию для фрактала tricorn.  
 \* Он принимает два двойных значения для действительной и мнимой частей комплексной плоскости  
 \* и возвращает количество итераций для соответствующей координаты. \*/* public int numIterations(double x, double y)  
 {  
 int iteration = 0;  
 double zreal = 0;  
 double zimaginary = 0;  
 while (iteration < MAX\_ITERATIONS && zreal \* zreal + zimaginary \* zimaginary < 4)  
 {  
 double zrealUpdated = zreal \* zreal - zimaginary \* zimaginary + x;  
 double zimaginaryUpdated = - 2 \* zreal \* zimaginary + y;  
 zreal = zrealUpdated;  
 zimaginary = zimaginaryUpdated;  
 iteration += 1;  
 }  
 // Если алгоритм дошел до значения MAX\_ITERATIONS нужно вернуть -1, чтобы показать, что точка не выходит за границы.  
 if (iteration == MAX\_ITERATIONS) return -1;  
 return iteration;  
 }  
 // Возращает имя фрактала  
 public String toString() {  
 return "Tricorn";  
 }

Рис.2- фрактал tricorn

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)*.
* package com.lab5;  
  import java.awt.geom.Rectangle2D;  
    
  // Этот класс является подклассом FractalGenerator. Он используется для вычисления фрактала BurningShip.  
  public class BurningShip extends FractalGenerator  
  {  
   // Константа для количества максимальных итераций.  
   public static final int MAX\_ITERATIONS = 2000;  
   */\*\* Этот метод позволяет генератору фракталов указать, какая часть комплексной плоскости наиболее интересна для фрактала.  
   \* Ему передается объект прямоугольника, и метод изменяет поля прямоугольника,  
   \* чтобы показать правильный начальный диапазон для фрактала.  
   \* Эта реализация устанавливает начальный диапазон в (-2 - 2.5i), (2 + 1.5i) или x = -2, y = -2.5, width = height = 4. \*/* public void getInitialRange(Rectangle2D.Double range)  
   {  
   range.x = -2;  
   range.y = -2.5;  
   range.width = 4;  
   range.height = 4;  
   }  
   */\*\* Этот метод реализует итерационную функцию для фрактала BurningShip.  
   \* Он принимает два двойных значения для действительной и мнимой частей комплексной плоскости  
   \* и возвращает количество итераций для соответствующей координаты. \*/* public int numIterations(double x, double y)  
   {  
   int iteration = 0;  
   double zreal = 0;  
   double zimaginary = 0;  
   while (iteration < MAX\_ITERATIONS && zreal \* zreal + zimaginary \* zimaginary < 4)  
   {  
   double zrealUpdated = zreal \* zreal - zimaginary \* zimaginary + x;  
   double zimaginaryUpdated = 2 \* Math.abs(zreal) \* Math.abs(zimaginary) + y;  
   zreal = zrealUpdated;  
   zimaginary = zimaginaryUpdated;  
   iteration += 1;  
   }  
   // Если алгоритм дошел до значения MAX\_ITERATIONS нужно вернуть -1, чтобы показать, что точка не выходит за границы.  
   if (iteration == MAX\_ITERATIONS) return -1;  
   return iteration;  
   }  
   // Возращает имя фрактала  
   public String toString() { return "Burning Ship"; }  
  }

Рис.3- фрактал Burning Ship

* 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».
* package com.lab5;  
  import java.awt.geom.Rectangle2D;  
    
  // Этот класс является подклассом FractalGenerator. Он используется для вычисления фрактала Мандельброта.  
  public class Mandelbrot extends FractalGenerator  
  {  
   // Константа для количества максимальных итераций.  
   public static final int MAX\_ITERATIONS = 2000;  
   */\*\* Этот метод позволяет генератору фракталов указать, какая часть комплексной плоскости наиболее интересна для фрактала.  
   \* Ему передается объект прямоугольника, и метод изменяет поля прямоугольника,  
   \* чтобы показать правильный начальный диапазон для фрактала.  
   \* Эта реализация устанавливает начальный диапазон в (-2 - 1,5i) - (1 + 1,5i) или x = -2, y = -1,5, width = height = 3. \*/* public void getInitialRange(Rectangle2D.Double range)  
   {  
   range.x = -2;  
   range.y = -1.5;  
   range.width = 3;  
   range.height = 3;  
   }  
   */\*\* Этот метод реализует итерационную функцию для фрактала Мандельброта.  
   \* Он принимает два двойных значения для действительной и мнимой частей комплексной плоскости  
   \* и возвращает количество итераций для соответствующей координаты. \*/* public int numIterations(double x, double y)  
   {  
   int iteration = 0;  
   double zreal = 0;  
   double zimaginary = 0;  
   while (iteration < MAX\_ITERATIONS && zreal \* zreal + zimaginary \* zimaginary < 4)  
   {  
   double zrealUpdated = zreal \* zreal - zimaginary \* zimaginary + x;  
   double zimaginaryUpdated = 2 \* zreal \* zimaginary + y;  
   zreal = zrealUpdated;  
   zimaginary = zimaginaryUpdated;  
   iteration += 1;  
   }  
   // Если алгоритм дошел до значения MAX\_ITERATIONS нужно вернуть -1, чтобы показать, что точка не выходит за границы.  
   if (iteration == MAX\_ITERATIONS) return -1;  
   return iteration;  
   }  
   // Возращает имя фрактала  
   public String toString() {  
   return "Mandelbrot";  
   }  
  }

Рис.4- фрактал Mandelbrot

* 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()) При этом не забудьте сбросить начальный диапазон и перерисовать фрактал!

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);

package com.lab5;  
import java.awt.\*;  
import javax.swing.\*;  
import java.awt.geom.Rectangle2D;  
import java.awt.event.\*;  
import javax.swing.JFileChooser.\*;  
import javax.swing.filechooser.\*;  
import javax.imageio.ImageIO.\*;  
import java.awt.image.\*;  
  
*/\*\* Этот класс позволяет исследовать различные части фрактала, создавая и показывая графический интерфейс Swing,  
 \* и обрабатывает события, вызванные различными взаимодействиями пользователя. \*/*public class FractalExplorer implements ItemListener  
{  
 private int displaySize; // Целое число «размер экрана», которое является шириной и высотой отображения в пикселях.  
 private JImageDisplay display; // Ссылка JImageDisplay, для обновления отображения в разных методах в процессе вычисления фрактала.  
 private FractalGenerator fractal; // Объект FractalGenerator, использующий ссылку базового класса для отображения других типов фракталов в будущем.  
 private Rectangle2D.Double range; // Объект Rectangle2D.Double, указывающий диапазона комплексной плоскости, которая выводится на экран.  
 // Конструктор, который принимает размер отображения, сохраняет его и инициализирует объекты диапазона и генератора фракталов.  
 public FractalExplorer(int size)  
 {  
 displaySize = size;  
 fractal = new Mandelbrot();  
 range = new Rectangle2D.Double();  
 fractal.getInitialRange(range);  
 display = new JImageDisplay(displaySize, displaySize);  
 }  
 // Этот метод инициализирует графический интерфейс Swing с помощью JFrame, содержащего объект JImageDisplay и кнопку для сброса отображения.  
 public void createAndShowGUI()  
 {  
 // Настройка фрейма на использование java.awt.BorderLayout для его содержимого.  
 display.setLayout(new BorderLayout());  
 JFrame myframe = new JFrame("Fractal Explorer");  
  
 JLabel header = new JLabel("Fractal:");  
 // Добавление объекта отображения изображения в позицию BorderLayout.CENTER.  
 myframe.add(display, BorderLayout.CENTER);  
 // Создание кнопок и панели выбора.  
 Choice Button1 = new Choice();  
 Button1.add("Mandelbrot");  
 Button1.add("Tricorn");  
 Button1.add("Burning Ship");  
 JPanel panel = new JPanel();  
 myframe.add(panel, BorderLayout.NORTH);  
 panel.add(header);  
 panel.add(Button1);  
 JButton resetButton = new JButton("Reset");  
 ResetHandler handler = new ResetHandler();  
 resetButton.addActionListener(handler);  
 JButton saveImage = new JButton("Save Image");  
 SaveHandler save = new SaveHandler();  
 saveImage.addActionListener(save);  
 JPanel down = new JPanel();  
 myframe.add(down, BorderLayout.SOUTH);  
 down.add(resetButton);  
 down.add(saveImage);  
 Button1.addItemListener(this);  
 MouseHandler click = new MouseHandler();  
 display.addMouseListener(click);  
 // Установка операции закрытия фрейма по умолчанию на «выход».  
 myframe.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);  
 // Размещение содержимого фрейма, чтобы оно было видимым и запрещение изменения размера окна.  
 myframe.pack();  
 myframe.setVisible(true);  
 myframe.setResizable(false);  
 }  
  
 */\*\* Приватный вспомогательный метод для отображения фрактала. Этот метод просматривает каждый пиксель на дисплее  
 \* и вычисляет количество итераций для соответствующих координат в области отображения фрактала.  
 \* Если количество итераций равно -1, цвет пикселя становится черным.  
 \* В противном случае выбираем значение на основе количества итераций, обновляем дисплей с использованием  
 \* цвета для каждого пикселя и перерисовываем JImageDisplay, когда все пиксели были нарисованы. \*/* private void drawFractal()  
 {  
 // Циклический просмотр каждого пикселя на дисплее  
 for (int x = 0; x < displaySize; x++)  
 for (int y = 0; y < displaySize; y++)  
 {  
 // Нахождение соответствующих координат xCoord и yCoord в области отображения фрактала.  
 double xCoord = fractal.getCoord(range.x,range.x + range.width, displaySize, x);  
 double yCoord = fractal.getCoord(range.y,range.y + range.height, displaySize, y);  
 // Вычисление количества итераций для координат в области отображения фрактала.  
 int iteration = fractal.numIterations(xCoord, yCoord);  
 if (iteration == -1) // Если количество итераций равно -1, устанавливаем пиксель черным.  
 display.drawPixel(x, y, 0);  
 else  
 {  
 // В противном случае выбираем значение оттенка в зависимости от количества итераций.  
 float hue = 0.7f + (float) iteration / 200f;  
 int rgbColor = Color.HSBtoRGB(hue, 1f, 1f);  
 display.drawPixel(x, y, rgbColor); // Обновление дисплея для каждого пикселя.  
 }  
 }  
 // Когда все пиксели нарисованы, перерисовываем JImageDisplay в соответствии с текущим содержимым его изображения.  
 display.repaint();  
 }  
 @Override  
 public void itemStateChanged(ItemEvent e)  
 {  
 if (e.getItem() == "Mandelbrot") fractal = new Mandelbrot();  
 if (e.getItem() == "Tricorn") fractal = new Tricorn();  
 if (e.getItem() == "Burning Ship") fractal = new BurningShip();  
 drawFractal();  
 }  
 private class SaveHandler implements ActionListener // Класс для сохранения  
 {  
 public void actionPerformed(ActionEvent e)  
 {  
  
 JFileChooser myFileChooser = new JFileChooser();  
 FileFilter extensionFilter = new FileNameExtensionFilter("PNG Images", "png");  
 myFileChooser.setFileFilter(extensionFilter);  
 myFileChooser.setAcceptAllFileFilterUsed(false);  
 int userSelection = myFileChooser.showSaveDialog(display);  
 if (userSelection == JFileChooser.APPROVE\_OPTION) {  
 java.io.File file = myFileChooser.getSelectedFile();  
  
 String file\_name = file.toString();  
 System.out.println(file\_name.lastIndexOf(".png"));  
 if (file\_name.lastIndexOf(".png") == -1)  
 {  
 file\_name += ".png";  
 file = new java.io.File(file\_name);  
 }  
 try {  
 BufferedImage displayImage = display.getImage();  
 javax.imageio.ImageIO.write(displayImage, "png", file);  
 }  
 catch (Exception exception) {  
 JOptionPane.showMessageDialog(  
 display, exception.getMessage(),  
 "Cannot Save Image", JOptionPane.ERROR\_MESSAGE);  
 }  
 }  
 }  
 }  
 private class ResetHandler implements ActionListener // Внутренний класс для обработки событий ActionListener от кнопки сброса.  
 {  
 // Обработчик сбрасывает диапазон до начального диапазона, заданного генератором, а затем рисует фрактал.  
 public void actionPerformed(ActionEvent e)  
 {  
 fractal.getInitialRange(range);  
 drawFractal();  
 }  
 }  
 private class MouseHandler extends MouseAdapter // Внутренний класс для обработки событий MouseListener с дисплея.  
 {  
 */\*\* Когда обработчик получает событие щелчка мыши, он сопоставляет пиксельные координаты щелчка  
 \* с областью отображаемого фрактала, а затем вызывает метод генератора correnterAndZoomRange() с координатами,  
 \* по которым щелкнули, и шкалой 0,5. \*/* @Override  
 public void mouseClicked(MouseEvent e)  
 {  
 // Получение координаты x области щелчка мыши.  
 int x = e.getX();  
 double xCoord = fractal.getCoord(range.x,range.x + range.width, displaySize, x);  
 // Получение координаты y области щелчка мыши.  
 int y = e.getY();  
 double yCoord = fractal.getCoord(range.y,range.y + range.height, displaySize, y);  
 // Вызов метода генератора RecenterAndZoomRange () с координатами, по которым был выполнен щелчок, и масштабом 0,5.  
 fractal.recenterAndZoomRange(range, xCoord, yCoord, 0.5);  
 drawFractal(); // Перерисовка фрактала после изменения отображаемой области.  
 }  
 }  
 */\*\* Статический метод main() для запуска FractalExplorer.  
 \* Инициализирует новый экземпляр FractalExplorer с размером отображения 600,  
 \* вызывает createAndShowGUI() в объекте проводника,  
 \* а затем вызывает drawFractal() в проводнике. \*/* public static void main(String[] args)  
 {  
 FractalExplorer displayExplorer = new FractalExplorer(600);  
 displayExplorer.createAndShowGUI();  
 displayExplorer.drawFractal();  
 }

Рис.5- FractalExplorer

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.

После того, как вы закончите реализацию этих функций, запустите. Теперь вы сможете исследовать различные фракталы, а также вы сможете

сохранять их на диск. Вы также можете проверить приложение на вывод сообщений об ошибках, попробуйте сохранить изображение в файл, который уже существует, но доступен только для чтения. Или вы можете попробовать сохранить файл с именем, которое является каталогом в целевой папке

package com.lab5;  
import javax.swing.\*;  
import java.awt.image.\*;  
import java.awt.\*;  
  
// Этот класс позволяет отображать фракталы.  
// Он является производным от javax.swing.JComponent.  
class JImageDisplay extends JComponent  
{  
 // Экземпляр буферизованного изображения.  
 // Управляет изображением, в содержимое которого мы можем писать.  
 private BufferedImage displayImage;  
 // Конструктор принимает целые значения ширины и высоты  
 // и инициализирует свой объект класса BufferedImage  
 // как новое изображение с этой шириной и высотой типа изображения TYPE\_INT\_RGB.  
 public JImageDisplay(int width, int height)  
 {  
 displayImage = new BufferedImage(width, height, BufferedImage.TYPE\_INT\_RGB);  
 // Вызов метода setPreferredSize() родительского класса с указаннной шириной и высотой.  
 Dimension imageDimension = new Dimension(width, height);  
 super.setPreferredSize(imageDimension);  
 }  
 // Реализация суперкласса paintComponent(g) вызывается для того,  
 // чтобы границы и объекты отображались правильно.  
 // Затем изображение отрисосывается в компонент.  
 @Override  
 public void paintComponent(Graphics g)  
 {  
 super.paintComponent(g);  
 g.drawImage(displayImage, 0, 0, displayImage.getWidth(), displayImage.getHeight(), null);  
 }  
 public void clearImage() // Устанавливает все пиксели в данных изображения в черный цвет.  
 {  
 int[] blankArray = new int[getWidth() \* getHeight()];  
 displayImage.setRGB(0, 0, getWidth(), getHeight(), blankArray, 0, 1);  
 }  
 // Устанавливает пиксель определенного цвета.  
 public void drawPixel(int x, int y, int rgbColor) { displayImage.setRGB(x, y, rgbColor); }  
 public BufferedImage getImage() {  
 return displayImage;  
 }  
}

Рис.6-JImageDisplay

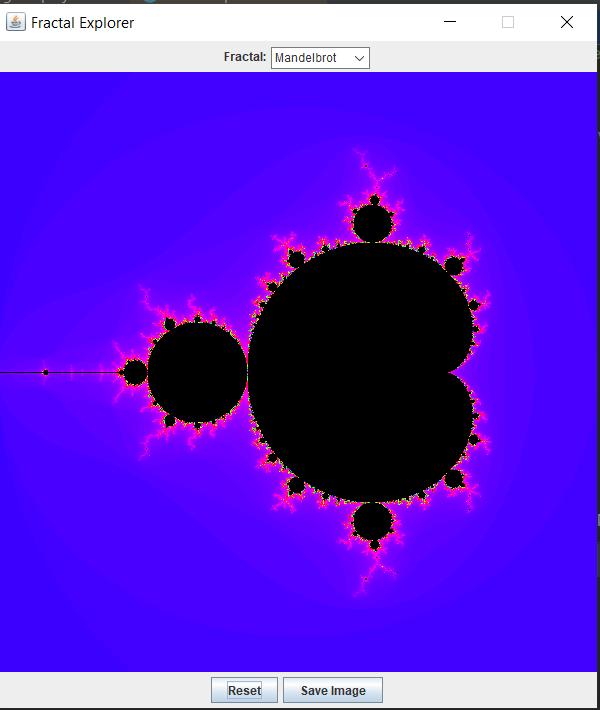


Рис.7 – Результат работы программы

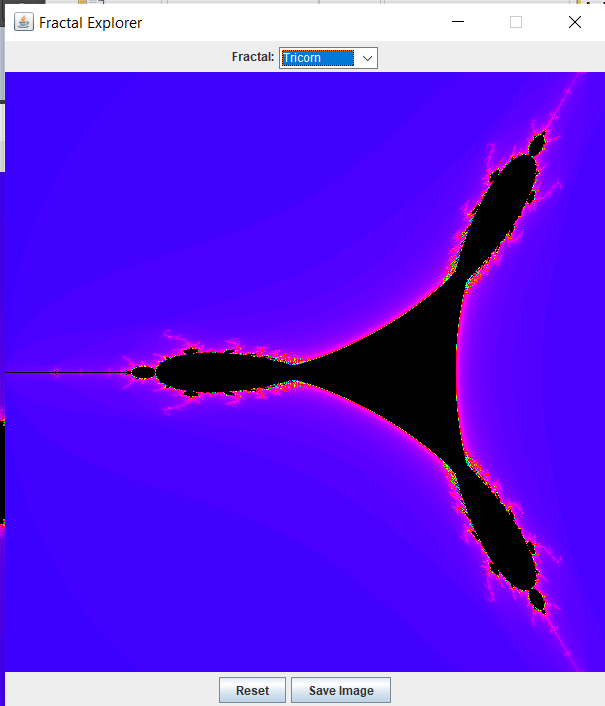


Рис.7 – Результат работы программы

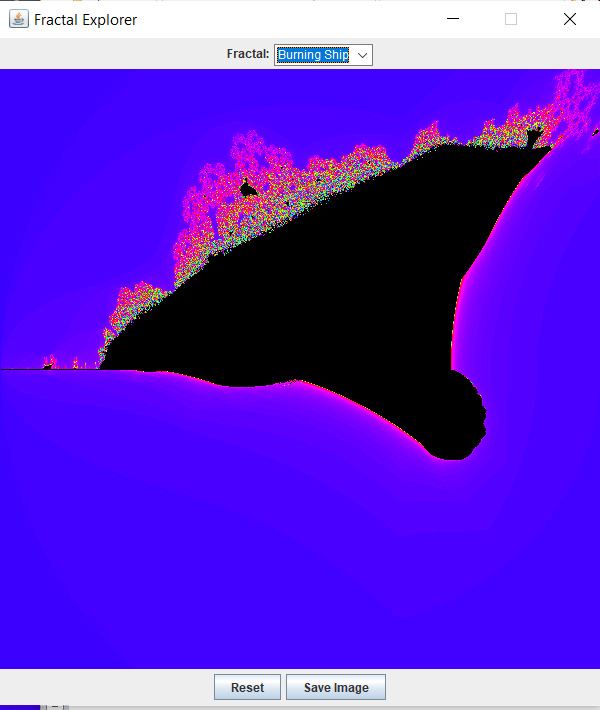


Рис.8 – Результат работы программы