**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 №6**

по дисциплине «Java-программирование»

на тему:

«Многопоточный генератор фракталов»

Выполнил: студ. гр. БПЗ1901

Неживлева Ксения

Проверил: Харрасов Камиль Раисович

Москва 2021

Цель работы:

В данной лабораторной работе необходимо будет реализовать возможность рисования фрактала с несколькими фоновыми потоками. Два преимущества данного подхода: первое - пользовательский интерфейс не будет зависать в процессе рисования нового фрактала, а второе - если у вас компьютер многоядерный, то процесс рисования будет намного быстрее.

Выполнение:

В данной лабораторной работе в основном необходимо будет работать в классе FractalExplorer. Часть кода будет новой, но некоторые части будут представлять из себя модифицированный код, который вы уже написали.

1. Создайте подкласс SwingWorker с именем FractalWorker, который будет внутренним классом FractalExplorer. Это необходимо для того, чтобы у него был доступ к нескольким внутренним членам FractalExplorer. Помните, что класс SwingWorker является универсальным, поэтому нужно указать параметры - можно просто указать Object для двух параметров, потому что в данной реализации эти параметры не будут использоваться. В результате у вас должна получиться следующая строчка кода:

private class FractalWorker extends SwingWorker<Object, Object>

1. Класс FractalWorker будет отвечать за вычисление значений цвета для одной строки фрактала, поэтому ему потребуются два поля: целочисленная y-координата вычисляемой строки, и массив чисел типа int для хранения вычисленных значений RGB для каждого пикселя в этой строке. Конструктор должен будет получать y-координату в качестве параметра и сохранять это. (На данном этапе не надо выделять память под целочисленный массив, так как он не потребуется, пока строка не будет вычислена.)
2. Метод doInBackground() вызывается в фоновом потоке и отвечает за выполнение длительной задачи. Поэтому в вашей реализации вам нужно будет взять часть кода из вашей предыдущей функции «draw fractal» и поместить ее в этот метод. Вместо того, чтобы рисовать изображение в окне, цикл должен будет сохранить каждое значение RGB в соответствующем элементе целочисленного массива. Вы не сможете изменять отображение из этого потока, потому что вы нарушите ограничения ограничения потоков Swing.
3. Вместо этого выделите память для массив целых чисел в начале реализации этого метода (массив должен быть достаточно большим для хранения целой строки значений цвета), а затем сохраните цвет каждого пикселя в этом массиве. Единственные различия между настоящим и предыдущим кодом в том, что вам нужно будет вычислить фрактал для указанной строки, и что вы на данном этапе не обновляете отображение.

Метод doInBackground() должен возвращать объект типа Object, так как это указано в объявлении SwingWorker <T, V>. Просто верните null.

1. Метод done() вызывается, когда фоновая задача завершена, и этот метод вызывается из потока обработки событий Swing. Это означает, что вы можете модифицировать компоненты Swing на ваш вкус. Поэтому в этом методе вы можете перебирать массив строк данных, рисуя пиксели, которые были вычислены в doInBackground ().

После того, как строка будет вычислена, вам нужно будет сообщить Swing, перерисовать часть изображения, которая была изменена.

**Игнорирование событий во время перерисовки**

 Создайте функцию void enableUI(boolean val), которая будет включать или отключать кнопки с выпадающим списком в пользовательском интерфейсе на основе указанного параметра. Для включения или отключения этих компонентов можно использовать метод Swing setEnabled(boolean). Убедитесь, что ваш метод обновляет состояние кнопки сохранения, кнопки сброса и выпадающего списка.

 Функция «draw fractal» должна сделать еще две вещи. Первая - она должна вызвать метод enableUI (false), чтобы отключить все элементы пользовательского интерфейса во время рисования. Вторая - она должна установить значение «rows remaining» равным общему количеству строк, которые нужно нарисовать. Эти действия должны быть сделаны перед выполнением каких-либо рабочих задач, иначе это может привести к некорректной работе алгоритма.

 В методе done(), уменьшите значение «rows remaining» на 1, как последний шаг данной операции. Затем, если после уменьшения значение «rows remaining» равно 0, вызовите метод enableUI (true).

 Наконец, измените реализацию mouse-listener для того, чтобы она сразу возвращалась в предыдущее состояние, если значение «rows remaining» не равно нулю. Другими словами, приложение будет реагировать на щелчки мышью, только в том случае, если больше нет строк, которые должны быть нарисованы. (Обратите внимание, что также не нужно вносить аналогичные

изменения в обработчике событий, потому что все эти компоненты будут отключены с помощью метода enableUI ().)

package com.lab6;  
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

package com.lab6;  
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.lab6;  
  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.lab6;  
  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

package com.lab6;  
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.  
 private int rows;  
 private JButton resetButton = new JButton("Reset");  
 private JButton saveImage = new JButton("Save Image");  
 private Choice choiceButton = new Choice();  
 // 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.  
 choiceButton.add("Mandelbrot");  
 choiceButton.add("Tricorn");  
 choiceButton.add("Burning Ship");  
 JPanel panel = new JPanel();  
 myframe.add(panel, BorderLayout.*NORTH*);  
 panel.add(header);  
 panel.add(choiceButton);  
 ResetHandler handler = new ResetHandler();  
 resetButton.addActionListener(handler);  
 SaveHandler save = new SaveHandler();  
 saveImage.addActionListener(save);  
 JPanel down = new JPanel();  
 myframe.add(down, BorderLayout.*SOUTH*);  
 down.add(resetButton);  
 down.add(saveImage);  
 choiceButton.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()  
 {  
 enableUI(false);  
 rows = displaySize;  
 // Циклически просмотреть каждую строку  
 for (int x = 0; x < displaySize; x++)  
 {  
 FractalWorker drawRow = new FractalWorker(x);  
 drawRow.execute();  
 }  
 }  
 @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)  
 {  
 if (rows != 0) return;  
 // 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.  
 }  
 }  
 private class FractalWorker extends SwingWorker<Object, Object>  
 {  
 int yCoordinate;  
 int[] computedRGBValues;  
 private FractalWorker(int row) {  
 yCoordinate = row;  
 }  
 protected Object doInBackground()  
 {  
 computedRGBValues = new int[displaySize];  
 for (int i = 0; i < computedRGBValues.length; i++)  
 {  
 double xCoord = fractal.*getCoord*(range.x,range.x + range.width, displaySize, i);  
 double yCoord = fractal.*getCoord*(range.y,range.y + range.height, displaySize, yCoordinate);  
 int iteration = fractal.numIterations(xCoord, yCoord);  
 if (iteration == -1) computedRGBValues[i] = 0;  
 else  
 {  
 float hue = 0.7f + (float) iteration / 200f;  
 int rgbColor = Color.*HSBtoRGB*(hue, 1f, 1f);  
 computedRGBValues[i] = rgbColor;  
 }  
 }  
 return null;  
 }  
 protected void done()  
 {  
 for (int i = 0; i < computedRGBValues.length; i++)  
 display.drawPixel(i, yCoordinate, computedRGBValues[i]);  
 display.repaint(0, 0, yCoordinate, displaySize, 1);  
 rows--;  
 if (rows == 0) enableUI(true);  
 }  
 }  
 private void enableUI(boolean val)  
 {  
 choiceButton.setEnabled(val);  
 resetButton.setEnabled(val);  
 saveImage.setEnabled(val);  
 }  
 */\*\* 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.

package com.lab6;  
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

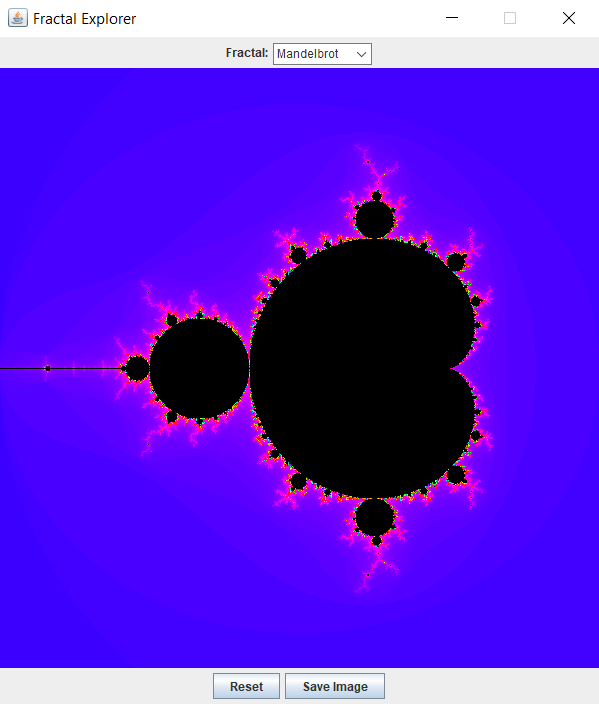


Рис.7 – Результат работы программы



Рис.7 – Результат работы программы

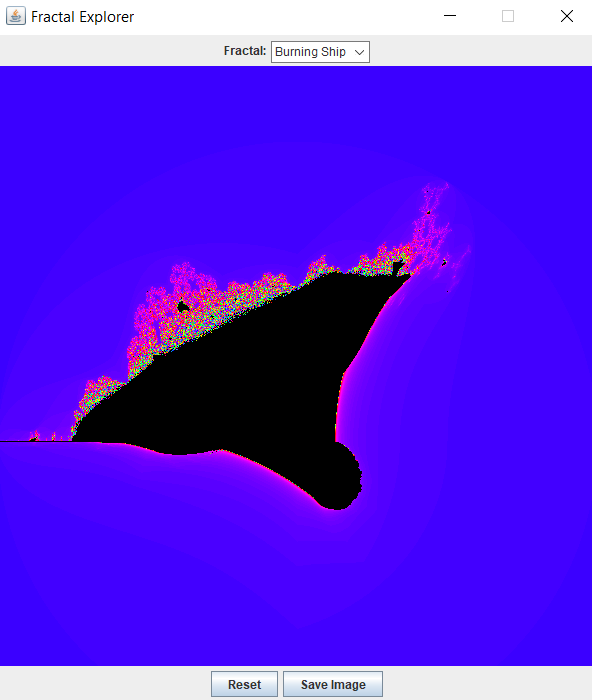


Рис.8 – Результат работы программы