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 №6

по дисциплине: «Многопоточный генератор 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 при помощи многопоточного генератора фракталов.

# 2. Задание на лабораторную работу

1) **Рисование в фоновом режиме**

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

2) Класс FractalWorker будет отвечать за вычисление значений цвета для одной строки фрактала, поэтому ему потребуются два поля: целочисленная yкоордината 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.)

3) 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.

4) 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 . Просто верните null.

5) 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. Поскольку вы изменили только одну строку, перерисовывать изображение целиком будет затратно, поэтому вы можете использовать метод JComponent.repaint(), который позволит вам указать область для перерисовки. У данного метода есть неиспользуемый параметр типа long, вы можете просто указать 0 для этого аргумента. В качестве остальных параметров укажите вычисленную строку, значения начала фрагмента для перерисовки (0, y) и конечные значения фрагмента (displaySize, 1). После того, как вы завершили класс для фоновой задачи, следующим шагом нужно будет привязать его к процессу рисования фракталов. Так как часть кода из функции «draw fractal» уже задействована в разрабатываемом классе, на данном этапе можно изменить функцию «draw fractal», а именно, для каждой строки в отображении создать отдельный рабочий объект, а затем вызвать для него метод execute (). Это действие запустит фоновый поток и запустит задачу в фоновом режиме. Помните, что класс FractalWorker отвечает за генерацию данных строки и за рисование этой строки, поэтому функция «draw fractal» должна быть простой. После завершения данной функции, вы сможете заметить отображения стало более быстрым, а пользовательский интерфейс стал более отзывчивым. Также вы сможете заметите одну проблему в вашем пользовательском интерфейсе - если вы нажмете на экран или на кнопку во время перерисовки, программа обработает это событие, хотя оно должно быть проигнорировано до завершения операции.

**2)** **Игнорирование событий во время перерисовки**

Самый простой способ решить проблему игнорирования событий во время перерисовки - отслеживать количество оставшихся строк, которые должны быть завершены, и игнорировать или отключать взаимодействие с пользователем до тех пор, пока не будут нарисованы все строки. Для нужно добавить поле «rows remaining» в класс Fractal Explorer и использовать его, чтобы узнать, когда будет завершена перерисовка. Чтение и запись этого значения будет происходить в потоке обработки событий, чтобы не было параллельного доступа к этому элементу. Если взаимодействие с ресурсом будет происходить только из одного потока, то не возникнет ошибок параллелизма. Для этого:

• 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 ().)

# 3. Ход лабораторной работы

## 3.1 Листинг программы

package lab6;  
  
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.\*;  
import javax.swing.filechooser.FileNameExtensionFilter;  
  
import lab6.fractal.\*;  
  
public class FractalExplorer {  
   
 */\*\*  
 размер экрана  
 \*/* private int dispSize;  
 private int rowsRemaining;  
 */\*\*  
 Ссылка для обновления отображения в разных методах в процессе вычисления фрактала  
 \*/* private JImageDisplay img;  
  
 */\*\*  
 \* Кнопка для сохранения  
 \*/* private JButton saveButton;  
  
 */\*\*  
 \* Кнопка для ресета  
 \*/* private JButton resetButton;  
 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.  
 \*/* private class FractalWorker extends SwingWorker<Object, Object> {  
  
  
 private int \_y;  
  
  
 private int[] rgbVals;  
  
 public FractalWorker(int y)  
 {  
 \_y = y;  
 }  
  
 */\*\*  
 \* Хранение каждое значение RGB в соответствующем элементе целочисленного массива  
 \*/* protected Object doInBackground()  
 {  
 rgbVals = new int[dispSize];  
  
 int numIters;  
 int rgbColor;  
  
 double yCoord = getFractlYcord(\_y);  
 double xCoord;  
 float hue;  
  
 for(int x = 0 ; x < dispSize ; ++x)  
 {  
 xCoord = getFractlXcord(x);  
  
 numIters = generetion.numIterations(xCoord, yCoord);  
 if(numIters < 0)  
 {  
 rgbColor = 0;  
 }  
 else  
 {  
 hue = 0.7f + numIters / 200f;  
 rgbColor = Color.*HSBtoRGB*(hue, 1f, 1f);  
 }  
  
 rgbVals[x] = rgbColor;  
 }  
  
 return null;  
 }  
  
 protected void done()  
 {  
 for(int x = 0 ; x < dispSize ; ++x)  
 {  
 img.drawPixel(x, \_y, rgbVals[x]);  
 }  
  
 img.repaint(0, 0, \_y, dispSize, 1);  
  
 if((--rowsRemaining) < 1)  
 {  
 enableUI(true);  
 }  
 }  
  
 }  
  
 //Игнорирование событий во время перерисовки  
 private void enableUI(boolean val)  
 {  
 fractalChos.setEnabled(val);  
  
 saveButton.setEnabled(val);  
 resetButton.setEnabled(val);  
 }  
  
  
 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  
 saveButton= new JButton("Save Image");  
 saveButton.setActionCommand("save");   
 saveButton.addActionListener(handler);  
 buttonsPanel.add(saveButton);  
   
 //Создаю кнопку для сброса дисплея  
 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()  
 {  
 this.enableUI(false);  
 rowsRemaining = dispSize;  
  
 for(int y = 0 ; y < dispSize ; ++y)  
 {  
 FractalWorker worker = new FractalWorker(y);  
 worker.execute();  
 }  
  
 img.repaint();  
 }  
   
 */\*\*  
 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  
 }   
}

## 3.2 Результат выполнения программы

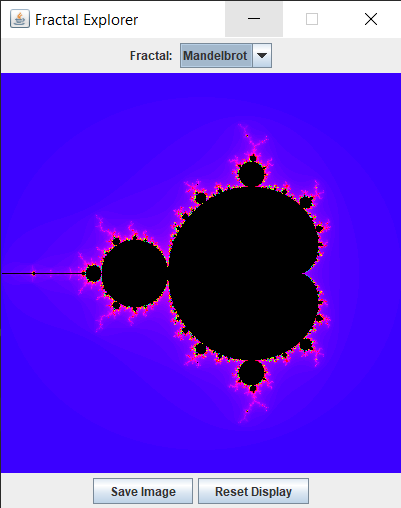


Рисунок 1 – результат выполнения

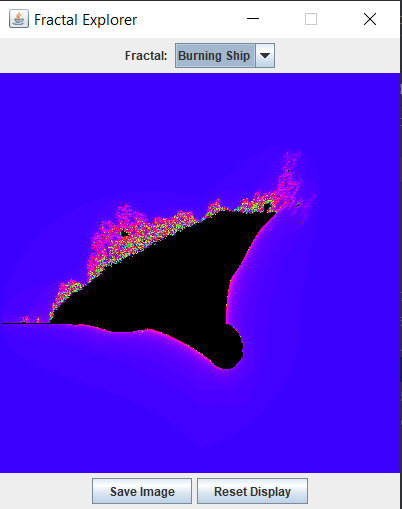


Рисунок 2 – результат выполнения

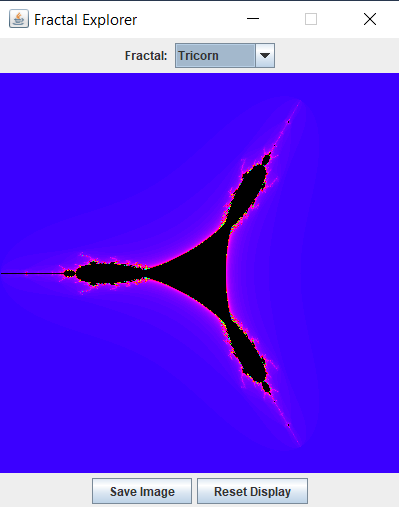


Рисунок 3 – результат выполнения

# 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