Министерство цифрового развития и массовых коммуникаций

Российской Федерации Ордена Трудового Красного Знамени

федеральное государственное образовательное бюджетное учреждение высшего профессионального образования

«Московский технический университет связи и информатики»

**Кафедра электроники**

**Отчет по лабораторной работе №6**

**По дисциплине «Кроссплатформенные технологии программирования» по теме:** **«Многопоточный генератор фракталов»**

Выполнил: студент

группы БСТ2001

Ибодуллоев У.Х.

Проверила:

Ст. преп. Полянцева К. А.

Москва

1. Цель работы:

В уже созданную ранее программу по отрисовке фракталов добавить функцию многопоточности с помощью фоновых потоков, а также игнорирование событий во время перерисовки.

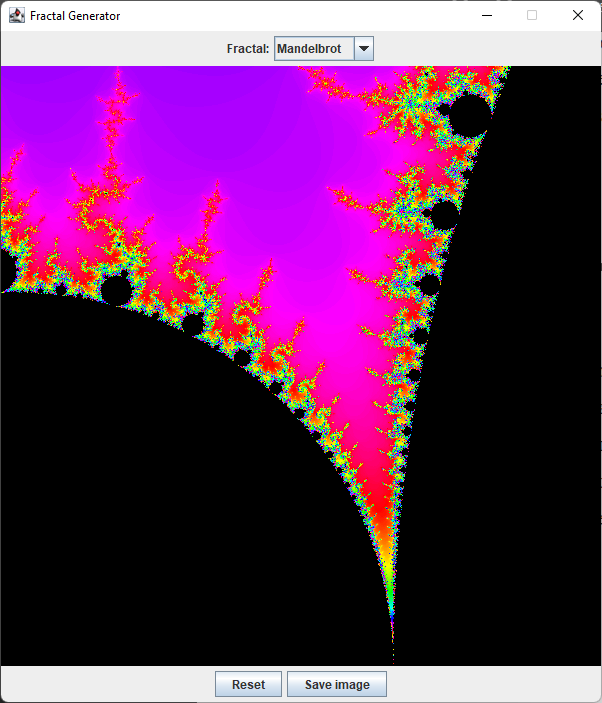
1. Задание:
2. Добавить новый класс FractalWorker для фоновой загрузки фрактала
3. Добавить новый класс EnableUI который будет включать или отключать кнопки с выпадающим списком
4. Код программы:

Ссылка на GitHub: <https://github.com/1Double/MTUCI/tree/main/Term_4/KTP/La>b6

1. Листинг файла FractalExplorer.java

import javax.swing.\*;  
import java.awt.\*;  
import java.io.IOException;  
import javax.imageio.ImageIO;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.awt.event.MouseAdapter;  
import java.awt.event.MouseEvent;  
import javax.swing.filechooser.FileNameExtensionFilter;  
import java.awt.geom.Rectangle2D;  
  
public class FractalExplorer {  
 private int displaySize;  
 private JImageDisplay imageDisplay;  
 private FractalGenerator fractalGenerator;  
 private Rectangle2D.Double range;  
 private JComboBox comboBox;  
 private int rowsRemaining;  
 private JButton buttonReset;  
 private JButton buttonSave;  
  
 private FractalExplorer (int displaySize) {  
 this.displaySize = displaySize;  
 this.fractalGenerator = new Mandelbrot();  
 this.range = new Rectangle2D.Double(0,0,0,0);  
 fractalGenerator.getInitialRange(this.range);  
 }  
  
 // точка входа  
 public static void main(String[] args) {  
 FractalExplorer fractalExplorer = new FractalExplorer(600);  
 fractalExplorer.setGUI();  
 fractalExplorer.drawFractal();  
 }  
  
 // задание интерфейса  
 public void setGUI() {  
 JFrame frame = new JFrame("Fractal Generator");  
 JPanel jPanel\_1 = new JPanel();  
 JPanel jPanel\_2 = new JPanel();  
 JLabel label = new JLabel("Fractal:");  
  
 imageDisplay = new JImageDisplay(displaySize, displaySize);  
 imageDisplay.addMouseListener(new MouseListener());  
  
 // выпадающий список  
 comboBox = new JComboBox();  
 comboBox.addItem(new Mandelbrot());  
 comboBox.addItem(new Tricorn());  
 comboBox.addItem(new BurningShip());  
 comboBox.addActionListener(new ActionHandler());  
  
 // кнопка reset  
 buttonReset = new JButton("Reset");  
 buttonReset.setActionCommand("Reset");  
 buttonReset.addActionListener(new ActionHandler());  
  
 // кнопка сохранить  
 buttonSave = new JButton("Save image");  
 buttonSave.setActionCommand("Save");  
 buttonSave.addActionListener(new ActionHandler());  
  
 jPanel\_1.add(label, BorderLayout.*CENTER*);  
 jPanel\_1.add(comboBox, BorderLayout.*CENTER*);  
 jPanel\_2.add(buttonReset, BorderLayout.*CENTER*);  
 jPanel\_2.add(buttonSave, BorderLayout.*CENTER*);  
  
 frame.setLayout(new BorderLayout());  
 frame.add(imageDisplay, BorderLayout.*CENTER*);  
 frame.add(jPanel\_1, BorderLayout.*NORTH*);  
 frame.add(jPanel\_2, BorderLayout.*SOUTH*);  
 frame.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
  
 frame.pack();  
 frame.setVisible(true);  
 frame.setResizable(false);  
 }  
  
 // отрисовка фрактала в JImageDisplay  
 private void drawFractal() {  
 // отключаем интерфейс на момент рисования  
 enableGUI(false);  
 rowsRemaining = displaySize;  
 for (int i = 0; i < displaySize; i++) {  
 FractalWorker drawRow = new FractalWorker(i);  
 drawRow.execute();  
 }  
 }  
  
 // включение - отключение gui  
 public void enableGUI(boolean b) {  
 buttonSave.setEnabled(b);  
 buttonReset.setEnabled(b);  
 comboBox.setEnabled(b);  
 }  
  
 // обработчик кнопок  
 public class ActionHandler implements ActionListener {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 if (e.getActionCommand().equals("Reset")) {  
 // перерисовка фрактала  
 fractalGenerator.getInitialRange(range);  
 drawFractal();  
 } else if (e.getActionCommand().equals("Save")) {  
 // сохранение  
 JFileChooser fileChooser = new JFileChooser();  
 FileNameExtensionFilter fileFilter = new FileNameExtensionFilter("PNG Images", "png");  
 fileChooser.setFileFilter(fileFilter);  
 fileChooser.setAcceptAllFileFilterUsed(false);  
 int t = fileChooser.showSaveDialog(imageDisplay);  
 if (t == JFileChooser.*APPROVE\_OPTION*) {  
 try {  
 ImageIO.*write*(imageDisplay.getImage(), "png", fileChooser.getSelectedFile());  
 } catch (NullPointerException | IOException ee) {  
 JOptionPane.*showMessageDialog*(imageDisplay, ee.getMessage(), "Cannot save image", JOptionPane.*ERROR\_MESSAGE*);  
 }  
 }  
 } else {  
 fractalGenerator = (FractalGenerator) comboBox.getSelectedItem();  
 range = new Rectangle2D.Double(0,0,0,0);  
 fractalGenerator.getInitialRange(range);  
 drawFractal();  
 }  
 }  
 }  
  
 public class MouseListener extends MouseAdapter {  
 @Override  
 public void mouseClicked(MouseEvent e) {  
 double x = FractalGenerator.*getCoord*(range.x, range.x + range.width, displaySize, e.getX());  
 double y = FractalGenerator.*getCoord*(range.y, range.y + range.width, displaySize, e.getY());  
 fractalGenerator.recenterAndZoomRange(range, x, y, 0.5);  
 drawFractal();  
 }  
 }  
  
 public class FractalWorker extends SwingWorker<Object, Object> {  
 private int y\_coord;  
 private int[] rgb;  
  
 public FractalWorker(int y\_coord) {  
 this.y\_coord = y\_coord;  
 }  
  
 @Override  
 protected Object doInBackground() throws Exception {  
 rgb = new int[displaySize];  
 for (int i = 0; i < displaySize; i++) {  
 int count = fractalGenerator.numIterations(FractalGenerator.*getCoord*(range.x, range.x + range.width, displaySize, i),  
 FractalGenerator.*getCoord*(range.y, range.y+range.width, displaySize, y\_coord));  
 if (count == -1)  
 rgb[i] = 0;  
 else {  
 double hue = 0.7f + (float) count / 200f;  
 int rgbColor = Color.*HSBtoRGB*((float) hue, 1f, 1f);  
 rgb[i] = rgbColor;  
 }  
 }  
 return null;  
 }  
  
 @Override  
 protected void done() {  
 for (int i = 0; i < displaySize; i++) {  
 imageDisplay.drawPixel(i, y\_coord, rgb[i]);  
 }  
 imageDisplay.repaint(0,0,y\_coord,displaySize,1);  
 rowsRemaining--;  
 if (rowsRemaining == 0)  
 enableGUI(true);  
 }  
 }  
}

1. Выполнение:



1. Выводы

В ходе работы я научился создавать многопоточные процессы, которые выполняются быстрее однопоточного.