**МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ**

**Ордена Трудового Красного Знамени**

**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Московский технический университет связи и информатики»**

Кафедра «Математическая кибернетика и информационные технологии»

**Отчет по лабораторной работе №6**

по дисциплине «Информационные технологии и программирование»

на тему:

Многопоточный генератор фракталов

Выполнил: студент группы БФИ2101

Иванов Глеб

Проверил преподаватель:

Мосева Марина Сергеевна

Москва, 2022

Оглавление

[Поставленная задача 3](#_Toc122872270)

[Скриншоты выполненной задачи 4](#_Toc122872271)

[Код 6](#_Toc122872272)

[Вывод 15](#_Toc122872273)

[Список источников 16](#_Toc122872274)

# 

**Поставленная задача**

1. Реализовать многопоточный генератор фракталов

**Скриншоты выполненной задачи**

****

Рисунок 1 – Отрисовка фрактала №1 (отрисовка происходит быстрее)

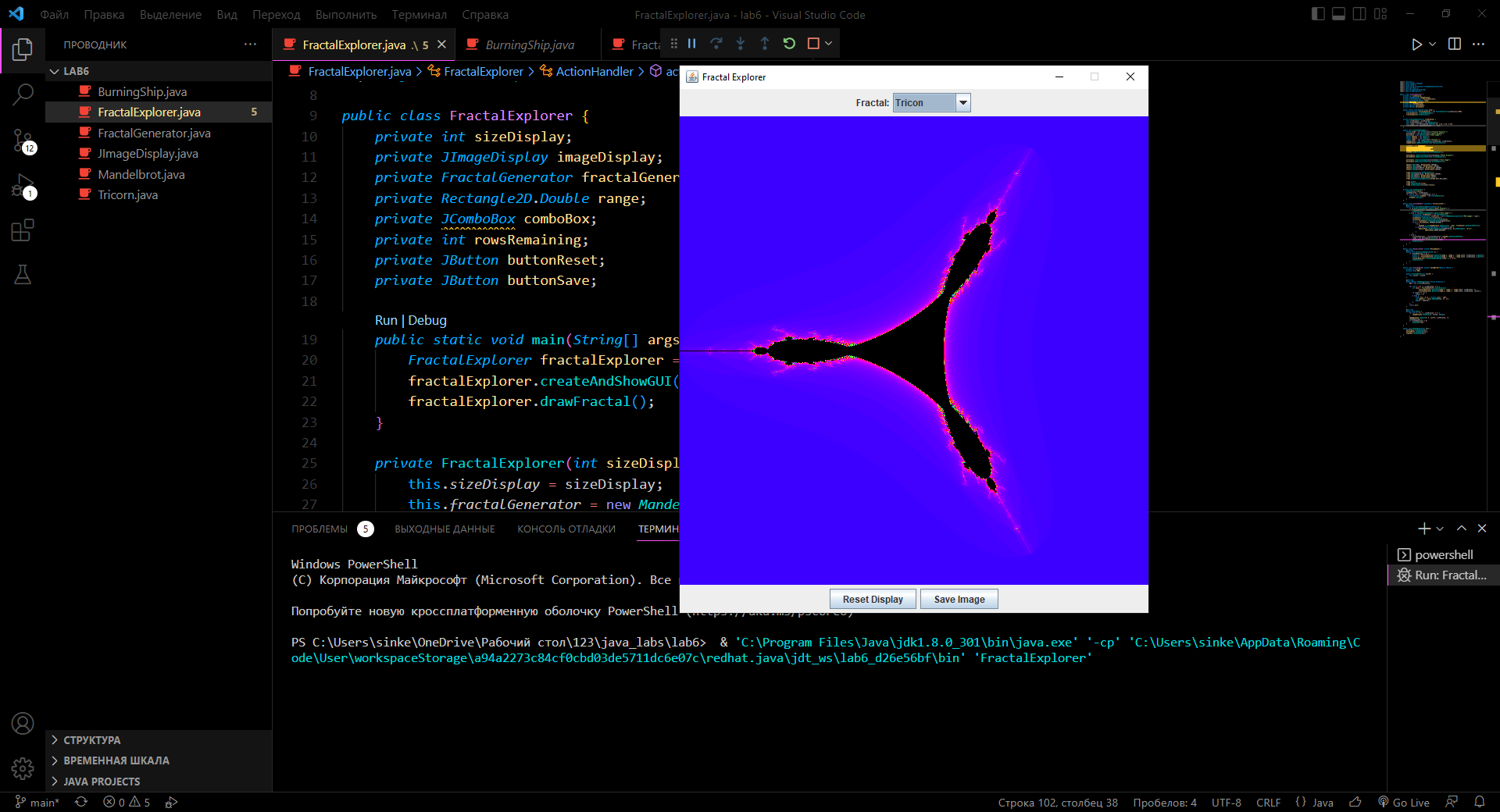


Рисунок 2 – Отрисовка фрактала №2 (отрисовка происходит быстрее)

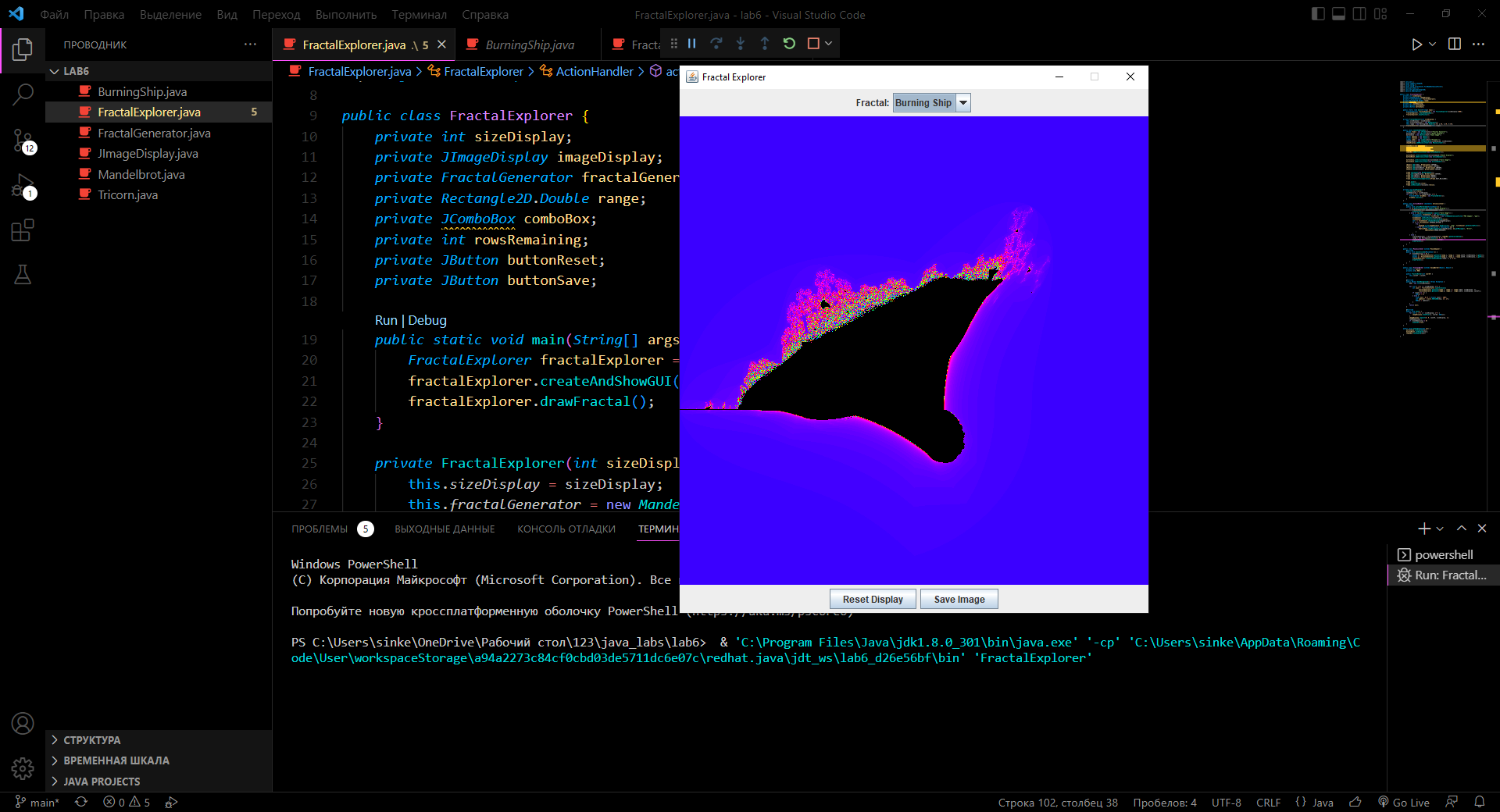


Рисунок 3 – Отрисовка фрактала №3 (отрисовка происходит быстрее)

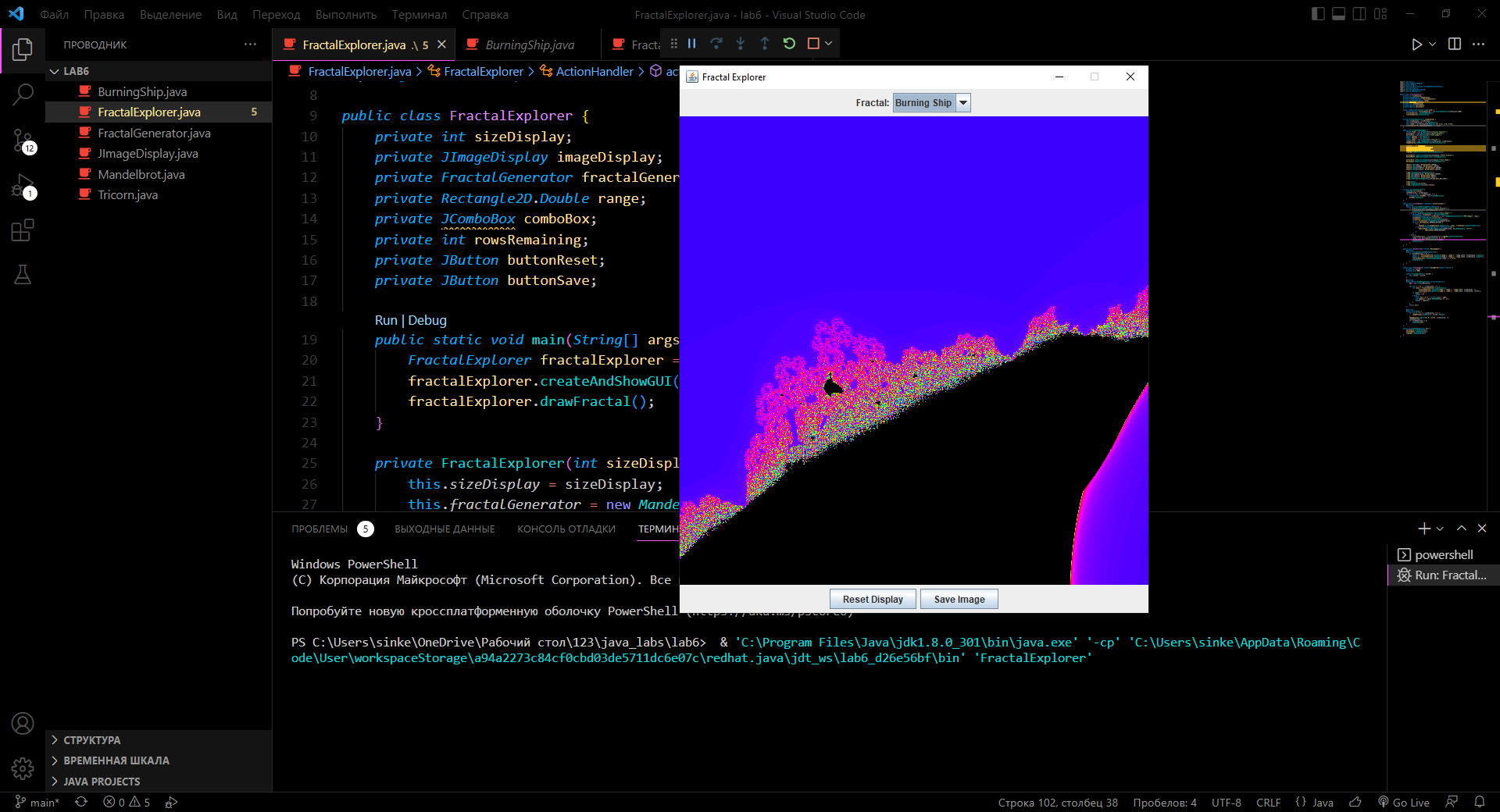


Рисунок 4 – Приближение отдельного участка изображения (приближение происходит быстрее)

**Код**

import *java*.*awt*.*geom*.*Rectangle2D*;

*public* *class* BurningShip *extends* FractalGenerator {

*public* *void* getInitialRange(*Rectangle2D*.*Double* range) {

        range.*x* = -2;

        range.*y* = -2.5;

        range.*height* = 4;

        range.*width* = 4;

    }

*public* *static* *final* *int* MAX\_ITERATIONS = 2000;

    @*Override*

*public* *int* numIterations(*double* x, *double* y) {

*double* i = y;

*double* x1 = x;

*int* count = 0;

        while (count < MAX\_ITERATIONS) {

            count++;

*double* a = x1 \* x1 - i \* i + x;

*double* b = Math.abs(2 \* x1 \* i) + y;

            x1 = a;

            i = b;

            if ((x1 \* x1 + i \* i) > 4)

                break;

        }

        if (count == MAX\_ITERATIONS)

            return -1;

        else

            return count;

    }

    @*Override*

*public* *String* toString() {

        return "Burning Ship";

    }

}

Код файла BurningShip.java

import *java*.*awt*.\*;

import *javax*.*imageio*.*ImageIO*;

import *javax*.*swing*.\*;

import *javax*.*swing*.*filechooser*.*FileNameExtensionFilter*;

import *java*.*awt*.*event*.\*;

import *java*.*awt*.*geom*.*Rectangle2D*;

import *java*.*io*.*IOException*;

*public* *class* FractalExplorer {

*private* *int* sizeDisplay;

*private* *JImageDisplay* imageDisplay;

*private* *FractalGenerator* fractalGenerator;

*private* *Rectangle2D*.*Double* range;

*private* *JComboBox* comboBox;

*private* *int* rowsRemaining;

*private* *JButton* buttonReset;

*private* *JButton* buttonSave;

*public* *static* *void* main(*String*[] args) {

*FractalExplorer* fractalExplorer = new *FractalExplorer*(600);

        fractalExplorer.createAndShowGUI();

        fractalExplorer.drawFractal();

    }

*private* FractalExplorer(*int* sizeDisplay) {

        this.*sizeDisplay* = sizeDisplay;

        this.*fractalGenerator* = new *Mandelbrot*();

        this.*range* = new Rectangle2D.Double(0, 0, 0, 0);

        fractalGenerator.getInitialRange(this.*range*);

    }

*public* *void* createAndShowGUI() {

*JFrame* frame = new *JFrame*("Fractal Explorer");

        buttonReset = new *JButton*("Reset Display");

        buttonSave = new *JButton*("Save Image");

*JPanel* jPanel1 = new *JPanel*();

*JPanel* jPanel2 = new *JPanel*();

*JLabel* label = new *JLabel*("Fractal:");

        imageDisplay = new *JImageDisplay*(*sizeDisplay*, *sizeDisplay*);

        imageDisplay.addMouseListener(new *MouseListener*());

        comboBox = new *JComboBox*();

        comboBox.addItem(new *Mandelbrot*());

        comboBox.addItem(new *Tricorn*());

        comboBox.addItem(new *BurningShip*());

        comboBox.addActionListener(new *ActionHandler*());

        buttonReset.setActionCommand("Reset Display");

        buttonReset.addActionListener(new *ActionHandler*());

        buttonSave.setActionCommand("Save Image");

        buttonSave.addActionListener(new *ActionHandler*());

        jPanel1.add(label, BorderLayout.*CENTER*);

        jPanel1.add(comboBox, BorderLayout.*CENTER*);

        jPanel2.add(buttonReset, BorderLayout.*CENTER*);

        jPanel2.add(buttonSave, BorderLayout.*CENTER*);

        frame.setLayout(new *BorderLayout*());

        frame.add(imageDisplay, BorderLayout.*CENTER*);

        frame.add(jPanel1, BorderLayout.*NORTH*);

        frame.add(jPanel2, BorderLayout.*SOUTH*);

        frame.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);

        frame.pack();

        frame.setVisible(true);

        frame.setResizable(false);

    }

*private* *void* drawFractal() {

*enableUI*(*false*);

        rowsRemaining = sizeDisplay;

        for (*int* i = 0; i < sizeDisplay; i++) {

*FractalWorker* drawRow = new *FractalWorker*(*i*);

            drawRow.execute();

        }

    }

*public* *class* ActionHandler *implements* ActionListener {

        @*Override*

*public* *void* actionPerformed(*ActionEvent* e) {

            if (e.getActionCommand().equals("Reset Display")) {

                fractalGenerator.getInitialRange(range);

*drawFractal*();

            } else if (e.getActionCommand().equals("Save Image")) {

*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.getPicture(), "png", fileChooser.getSelectedFile());

                    } catch (*NullPointerException* | *IOException* ee) {

                        JOptionPane.showMessageDialog(imageDisplay, ee.getMessage(), "Error",

                                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) {

            if (rowsRemaining == 0) {

*double* x = FractalGenerator.getCoord(range.*x*, range.*x* + range.*width*, sizeDisplay, e.getX());

*double* y = FractalGenerator.getCoord(range.*y*, range.*y* + range.*width*, sizeDisplay, e.getY());

                fractalGenerator.recenterAndZoomRange(range, x, y, 0.5);

*drawFractal*();

            }

        }

    }

*public* *class* FractalWorker *extends* SwingWorker<*Object*, *Object*> {

*private* *int* coordY;

*private* *int*[] RGB;

*public* FractalWorker(*int* coordY) {

            this.*coordY* = coordY;

        }

        @*Override*

*public* *Object* doInBackground() *throws* *Exception* {

            RGB = new *int*[sizeDisplay];

            for (*int* x = 0; x < sizeDisplay; x++) {

*int* count = fractalGenerator.numIterations(

                        FractalGenerator.getCoord(range.*x*, range.*x* + range.*width*, sizeDisplay, x),

                        FractalGenerator.getCoord(range.*y*, range.*y* + range.*width*, sizeDisplay, coordY));

                if (count == -1) {

*RGB*[x] = 0;

                } else {

*float* hue = 0.7f + (*float*) count / 200f;

*int* rgbColor = Color.HSBtoRGB(hue, 1f, 1f);

*RGB*[x] = rgbColor;

                }

            }

            return null;

        }

        @*Override*

*public* *void* done() {

            for (*int* x = 0; x < sizeDisplay; x++) {

                imageDisplay.drawPixel(x, coordY, *RGB*[x]);

            }

            imageDisplay.repaint(0, 0, coordY, sizeDisplay, 1);

            rowsRemaining--;

            if (rowsRemaining == 0)

*enableUI*(*true*);

        }

    }

*public* *void* enableUI(*boolean* val) {

        buttonSave.setEnabled(val);

        buttonReset.setEnabled(val);

        comboBox.setEnabled(val);

    }

}

Код файла FractalExplorer.java

import *java*.*awt*.*geom*.*Rectangle2D*;

*public* *abstract* *class* FractalGenerator {

*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);

    }

*public* *abstract* *void* getInitialRange(*Rectangle2D*.*Double* range);

*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;

    }

*public* *abstract* *int* numIterations(*double* x, *double* y);

}

Код файла FractalGenerator.java

import *javax*.*swing*.\*;

import *java*.*awt*.\*;

import *java*.*awt*.*image*.*BufferedImage*;

*public* *class* JImageDisplay *extends* JComponent

{

*private* *BufferedImage* picture;

*public* JImageDisplay(*int* width, *int* height)

    {

        picture = new *BufferedImage*(*width*, *height*,BufferedImage.*TYPE\_INT\_RGB*);

*Dimension* dimension = new *Dimension*(*width*, *height*);

        super.setPreferredSize(dimension);

    }

*public* *void* paintComponent(*Graphics* g)

    {

        super.paintComponent(g);

        g.drawImage(picture, 0, 0, picture.getWidth(), picture.getHeight(), null);

    }

*public* *void* clearImage()

    {

        for (*int* i = 0; i < picture.getWidth(); i++)

        {

            for (*int* j = 0; j < picture.getHeight(); j++)

            {

*drawPixel*(*i*, *j*,0);

            }

        }

    }

*public* *void* drawPixel(*int* x, *int* y, *int* rgbColor)

    {

        picture.setRGB(x, y, rgbColor);

    }

*public* *BufferedImage* getPicture()

    {

        return picture;

    }

}

Код файла JImageDisplay.java

import *java*.*awt*.*geom*.*Rectangle2D*;

*public* *class* Mandelbrot *extends* FractalGenerator {

*public* *void* getInitialRange(*Rectangle2D*.*Double* range) {

        range.*x* = -2;

        range.*y* = -1.5;

        range.*height* = 3;

        range.*width* = 3;

    }

*public* *static* *final* *int* MAX\_ITERATIONS = 2000;

    @*Override*

*public* *int* numIterations(*double* x, *double* y) {

*double* i = y;

*double* x1 = x;

*int* count = 0;

        while (count < MAX\_ITERATIONS) {

            count++;

*double* a = x1 \* x1 - i \* i + x;

*double* b = 2 \* x1 \* i + y;

            x1 = a;

            i = b;

            if (x1 \* x1 + i \* i > 4)

                break;

        }

        if (count == MAX\_ITERATIONS)

            return -1;

        else

            return count;

    }

    @*Override*

*public* *String* toString() {

        return "Mandelbrot";

    }

}

Код файла Mandelbrot.java

import *java*.*awt*.*geom*.*Rectangle2D*;

*public* *class* Tricorn *extends* FractalGenerator {

*public* *void* getInitialRange(*Rectangle2D*.*Double* range) {

        range.*x* = -2;

        range.*y* = -2;

        range.*height* = 4;

        range.*width* = 4;

    }

*public* *static* *final* *int* MAX\_ITERATIONS = 2000;

    @*Override*

*public* *int* numIterations(*double* x, *double* y) {

*double* i = y;

*double* x1 = x;

*int* count = 0;

        while (count < MAX\_ITERATIONS) {

            count++;

*double* a = x1 \* x1 - i \* i + x;

*double* b = (-2) \* x1 \* i + y;

            x1 = a;

            i = b;

            if (x1 \* x1 + i \* i > 4)

                break;

        }

        if (count == MAX\_ITERATIONS)

            return -1;

        else

            return count;

    }

    @*Override*

*public* *String* toString() {

        return "Tricon";

    }

}

Код файла Tricorn.java

**Вывод**

В ходе данной лабораторной работы я реализовал многопоточность и ускорил работу нашего пользовательского интерфейса.

**Список источников**

1. Камаев В.А., Костерин В.В. Технологии программирования. М.: Высшая школа, 2006.

2. Жоголев Е.А.Технология программирования. – М.: Научный мир, 2004.