**Федеральное агентство связи**

Ордена Трудового Красного Знамени

федеральное государственное бюджетное образовательное учреждение высшего образования   
«Московский технический университет связи и информатики»

Кафедра «Информатика»

Лабораторная работа №6: Многопоточный генератор фракталов

по дисциплине «Кроссплатформенные технологии программирования»

В данной лабораторной работе необходимо будет реализовать возможность рисования фрактала с несколькими фоновыми потоками.

Выполнил: студент

группы БСТ1803

Шевцов Никита

Вариант №24

Москва 2020

**Содержание**

[1 Задание на разработку программы 2](#_Toc53411831)

[2 Разработанные функции и классы 9](#_Toc53411832)

[2.1 Класс FractalExplorer 9](#_Toc53411833)

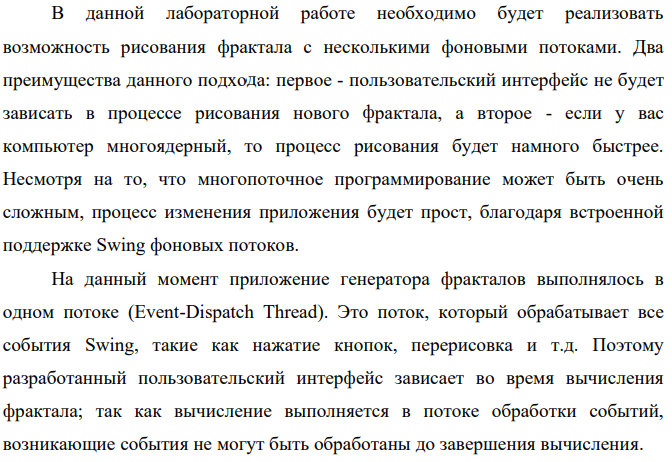
[2.1.1 Внутренний класс FractalWorker подкласс SwingWorker 9](#_Toc53411834)

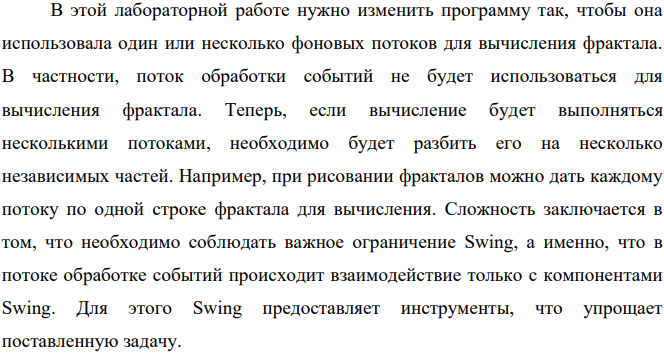
[2.2 Метод void enableUI 9](#_Toc53411835)

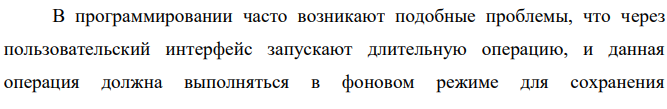
[3 Код программы 10](#_Toc53411836)

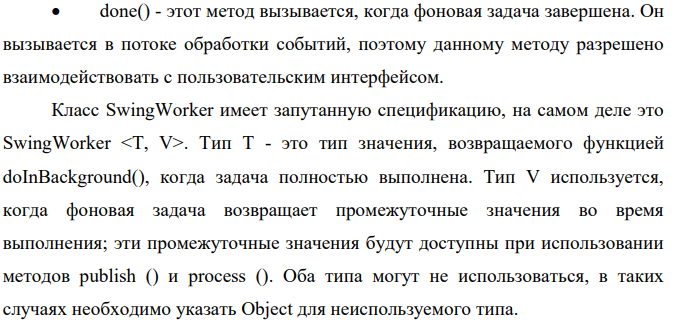
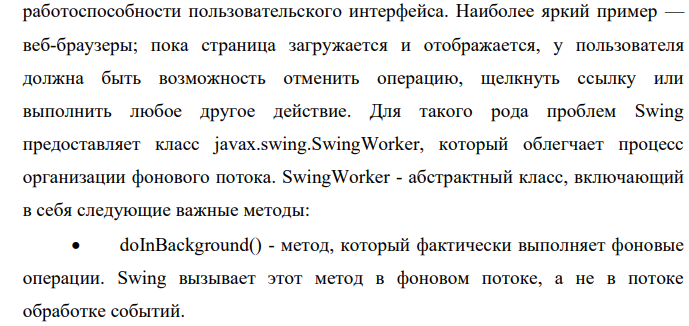
[4 Результат работы программы 15](#_Toc53411837)

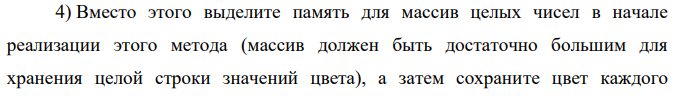
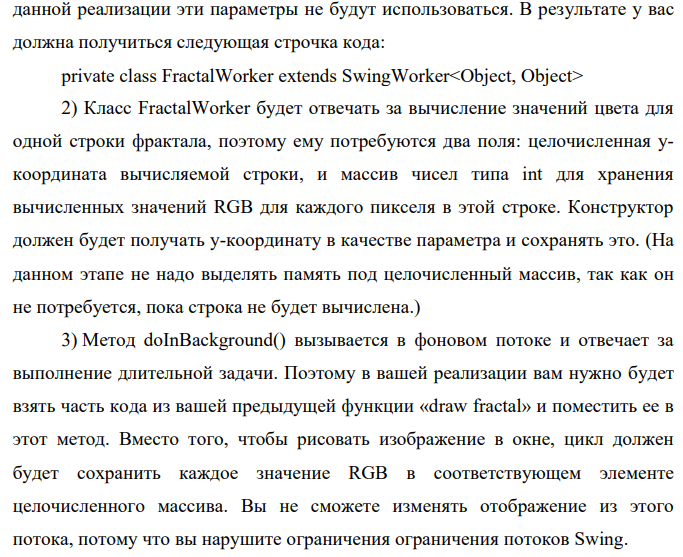
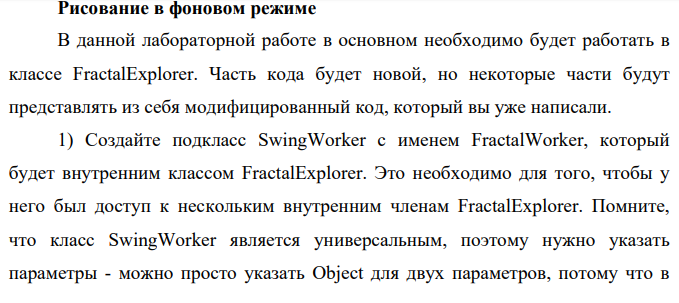
# Задание на разработку программы

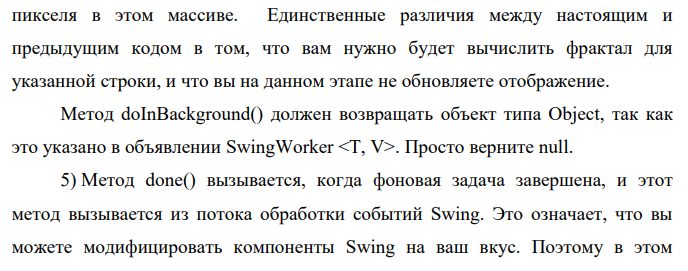


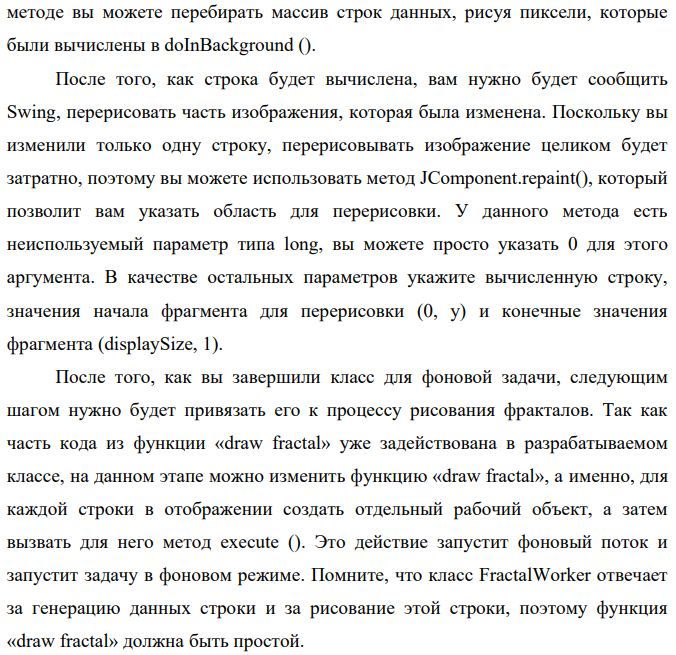


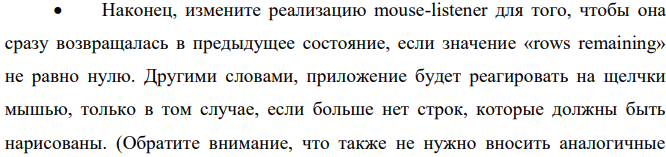
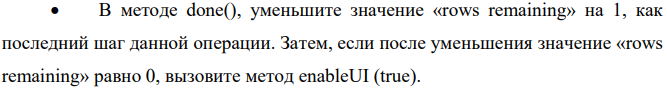
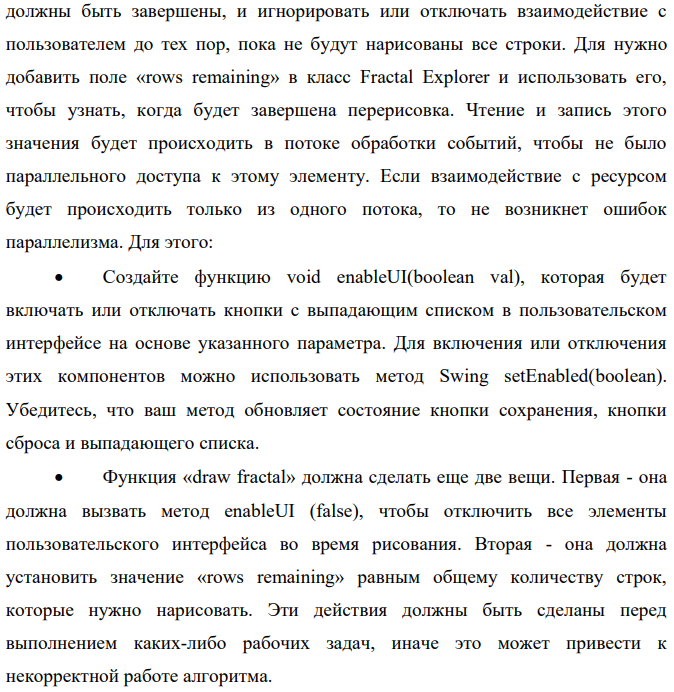
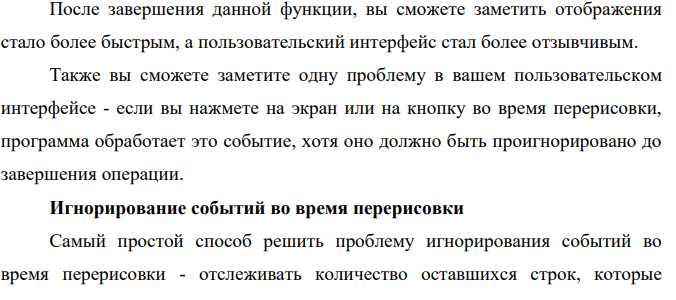


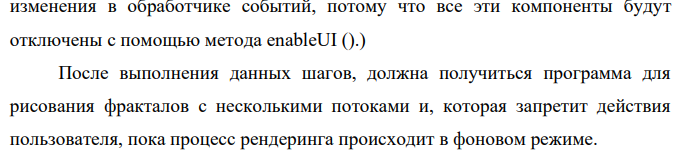












# Разработанные функции и классы

## Класс FractalExplorer

Класс отрисовывающий графический интерфейс Swing.

* + 1. Внутренний класс FractalWorker подкласс SwingWorker

Класс осуществляющий отрисовку строки изображения, необходим для многопоточной обработки изображения

* + - 1. Метод Object doInBackground()

Метод обработки в потоке, вычисляет массив цветов для строки

* + - 1. Метод void done()

Метод вызываемый, при окончании обработки в потоке, заполняет изображение вычисленными цветами и перерисовывает его.

* 1. Метод void enableUI

Метод блокировки/разблокировки интерфейса.

# Код программы

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

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

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

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

**import** java.io.File;

**import** java.io.IOException;

**import** java.util.HashMap;

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

**import** javax.imageio.ImageIO;

/\*\*

\* Приложение для отображения и исследования фракталов

\* **@author** niksh

\* **@version** 1.2

\*/

**public** **class** FractalExplorer **extends** JFrame {

**private** **static** **final** **long** ***serialVersionUID*** = -8728543713519701422L;

/\*\* Размер экрана \*/

**private** **int** windowSize;

/\*\* Элемент управления изображением \*/

**private** JImageDisplay image;

/\*\* Основа генерации фракталов \*/

**private** FractalGenerator fgen;

/\*\* Диапазон комплексной плоскости для отображения \*/

**private** Rectangle2D.Double range;

/\*\* Хеш-таблица компонентов окна \*/

**private** HashMap<Integer,Component> componentMap;

/\*\* Кол-во оставшишся для отрисовки рядов изображения \*/

**private** **int** rowsRemainig;

/\*\*

\* Конструктор - инициализирует компоненты отображения фрактала

\* **@param** size - размер изображения

\*/

**public** FractalExplorer(**int** size) {

**this**.windowSize = size;

**this**.image = **new** JImageDisplay(**this**.windowSize, **this**.windowSize);

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

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

fgen.getInitialRange(range);

}

/\*\*

\* Конструктор - инициализирует компоненты отображения фрактала со стандартным размером 500px

\*/

**public** FractalExplorer() {

**this**(JImageDisplay.***DEFAULT\_SIZE***);

}

/\*\*

\* Метод-инициализатор для GUI

\*/

**public** **void** createAndShowGUI() {

// Изображение фрактала

**this**.image.addMouseListener(**new** mouse1\_Click());

**this**.add(**this**.image, BorderLayout.***CENTER***);

ActionListener actions = **new** button\_Click();

// Кнопка сброса

JButton reset = **new** JButton("Сброс");

reset.setActionCommand("reset");

reset.addActionListener(actions);

// Кнопка сохранения

JButton save = **new** JButton("Сохранить");

save.setActionCommand("save");

save.addActionListener(actions);

// Контейнер кнопок

JPanel bottom = **new** JPanel();

bottom.add(save);

bottom.add(reset);

**this**.add(bottom, BorderLayout.***SOUTH***);

// Выпадающий лист фракталов

JComboBox<FractalGenerator> list = **new** JComboBox<FractalGenerator>();

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

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

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

list.setActionCommand("change");

list.addActionListener(actions);

JLabel boxLabel = **new** JLabel("Выбор фрактала");

// Контейнер листа и имени

JPanel top = **new** JPanel();

top.add(boxLabel);

top.add(list);

**this**.add(top, BorderLayout.***NORTH***);

// Настройка окна

**this**.setTitle("Fractal Explorer");

**this**.setDefaultCloseOperation(***EXIT\_ON\_CLOSE***);

**this**.pack();

**this**.setVisible(**true**);

**this**.setResizable(**true**);

// Добавление необходимых компонентов в HashMap

**this**.componentMap = **new** HashMap<Integer,Component>();

**this**.componentMap.put(0, save);

**this**.componentMap.put(1, reset);

**this**.componentMap.put(2, list);

**this**.componentMap.put(3, image);

}

/\*\*

\* Метод, отрисовывающий фрактал на экран

\*/

**private** **void** drawFractal() {

**this**.rowsRemainig = **this**.windowSize;

**this**.enableUI(**false**);

**for** (**int** y = 0; y < **this**.windowSize; y++) {

FractalWorker row = **new** FractalWorker(y);

row.execute();

}

}

/\*\*

\* Метод - включает или выключает заданные в {@link FractalExplorer#componentMap} элементы GUI

\* **@param** toggle - <code>true</code> включить, <code>false</code> выключить

\*/

**private** **void** enableUI(**boolean** toggle) {

**for** (**int** i = 0; i < componentMap.size(); i++) {

componentMap.get(i).setEnabled(toggle);

}

}

/\*\*

\* Класс, реализующий многопоточную отрисовку фрактала через Swing

\*/

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

/\*\* <code>y</code> координата фрактала \*/

**private** **int** yCoord;

/\*\* Массив цветов выбранной строки фрактала \*/

**public** **int**[] xRGB;

/\*\*

\* Конструктор - создание объекта для вычисления строки

\* **@param** y - строка для вычисления

\*/

**public** FractalWorker(**int** y) {

**this**.yCoord = y;

}

/\*\*

\* Фоновое вычисление одного ряда изображения фрактала

\*/

@Override

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

**this**.xRGB = **new** **int**[windowSize];

**double** imag = FractalGenerator.*getCoord*(range.y, range.y + range.height, windowSize, yCoord);

**int** iter;

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

**double** real = FractalGenerator.*getCoord*(range.x, range.x + range.width, windowSize, x);

iter = fgen.numIterations(real, imag);

**if** (iter == -1) xRGB[x] = 0;

**else** {

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

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

xRGB[x] = rgbColor;

}

}

**return** **null**;

}

/\*\*

\* Отрисовка полученной строки по завершению вычислений

\*/

@Override

**protected** **void** done() {

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

image.drawPixel(x, yCoord, xRGB[x]);

}

image.repaint(0, yCoord, windowSize, 1);

rowsRemainig -= 1;

**if** (rowsRemainig == 0) enableUI(**true**);

**super**.done();

}

}

/\*\*

\* Класс событий для элементов GUI

\*/

**private** **class** button\_Click **implements** ActionListener {

@Override

**public** **void** actionPerformed(ActionEvent arg0) {

String command = arg0.getActionCommand();

**switch** (command) {

// Кнопка сохранения

**case** ("save"):

JFileChooser dialog = **new** JFileChooser();

dialog.setLocale(getLocale());

FileNameExtensionFilter pngFilter = **new** FileNameExtensionFilter("PNG files","png");

dialog.setFileFilter(pngFilter);

dialog.setAcceptAllFileFilterUsed(**false**);

**int** result = dialog.showSaveDialog(image);

**if** (result == JFileChooser.***APPROVE\_OPTION***) {

**try** {

ImageIO.*write*(image.bimage,"png",**new** File(dialog.getSelectedFile()+".png"));

} **catch** (IOException e) {

JOptionPane.*showMessageDialog*(dialog, e.getLocalizedMessage(), "Save error", JOptionPane.***ERROR\_MESSAGE***);

e.printStackTrace();

}

}

**break**;

//Кнопка сброса

**case** ("reset"):

fgen.getInitialRange(range);

drawFractal();

**break**;

// Лист фракталов

**case** ("change"):

@SuppressWarnings("unchecked") JComboBox<FractalGenerator> list = (JComboBox<FractalGenerator>) arg0.getSource();

fgen = (FractalGenerator) list.getSelectedItem();

fgen.getInitialRange(range);

drawFractal();

**break**;

}

}

}

/\*\*

\* Класс событий для мыши (принадлежит {@link FractalExplorer#image})

\*/

**private** **class** mouse1\_Click **extends** MouseAdapter {

// Приблизить фрактал в точку, в которой находился курсор на момент отпускания ЛКМ

@Override

**public** **void** mouseReleased(MouseEvent arg0) {

**if** (image.isEnabled()) {

Point loc = arg0.getPoint();

**double** x = FractalGenerator.*getCoord*(range.x, range.x + range.width, windowSize, loc.x);

**double** y = FractalGenerator.*getCoord*(range.y, range.y + range.height, windowSize, loc.y);

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

drawFractal();

}

}

}

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

FractalExplorer app = **new** FractalExplorer(800);

app.createAndShowGUI();

app.drawFractal();

}

}

# Результат работы программы

