**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

Корнеев Александр

Вариант 12

Москва 2020

**Содержание**

[1 Задание на разработку программы 2](#_Toc41820380)

[2 Разработанные функции и классы 9](#_Toc41820381)

[2.1 Класс FractalExplorer 9](#_Toc41820382)

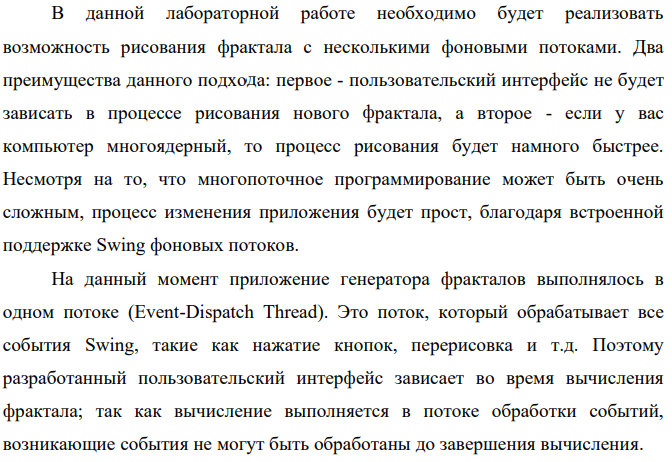
[2.1.1 Внутренний класс FractalWorker подкласс SwingWorker 9](#_Toc41820383)

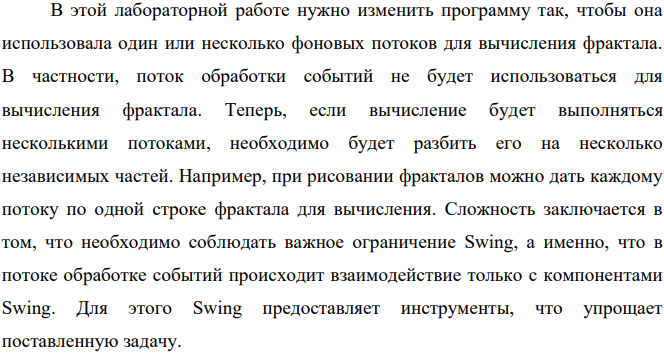
[2.2 Метод void enableUI 9](#_Toc41820384)

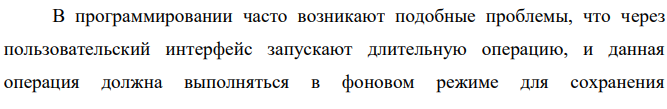
[3 Код программы 10](#_Toc41820385)

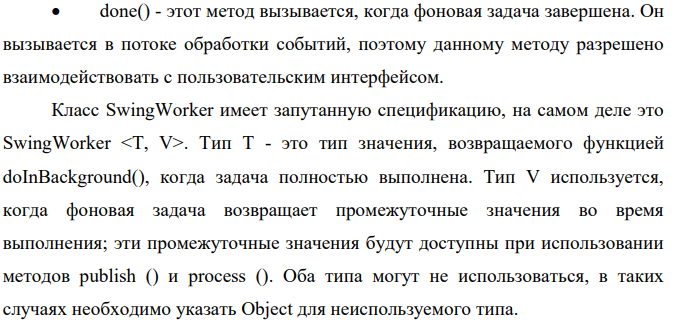
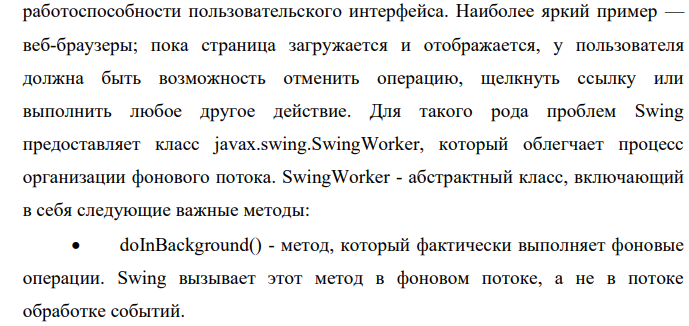
[4 Результат работы программы 15](#_Toc41820386)

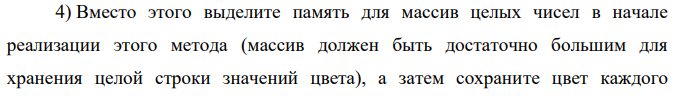
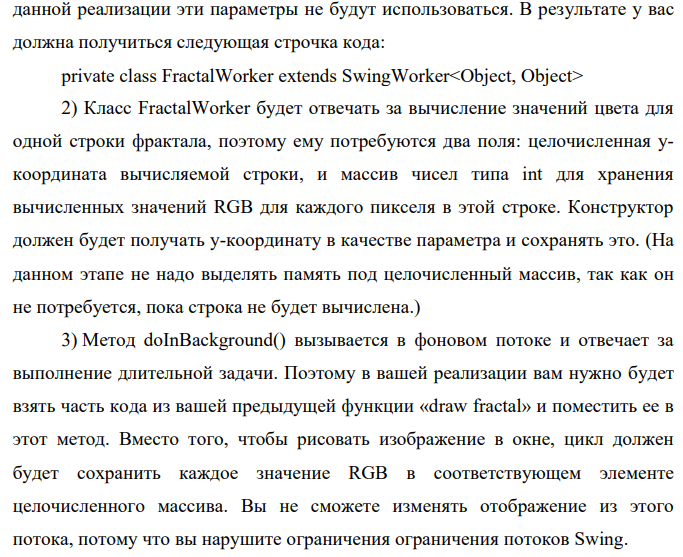
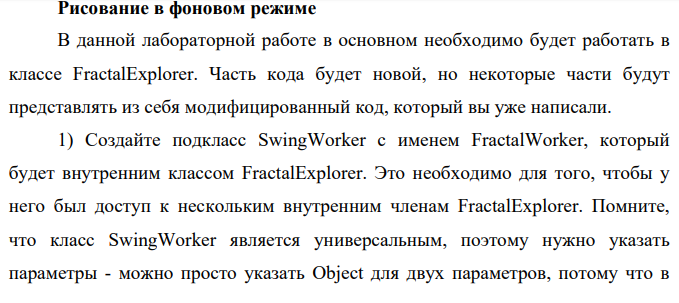
# Задание на разработку программы

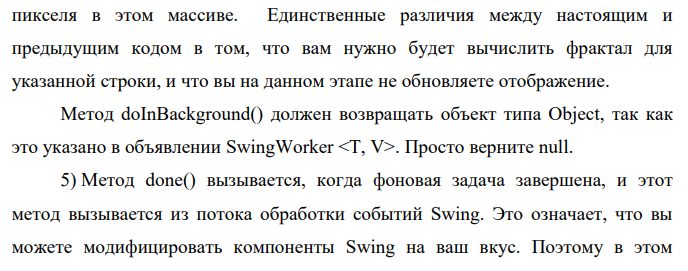


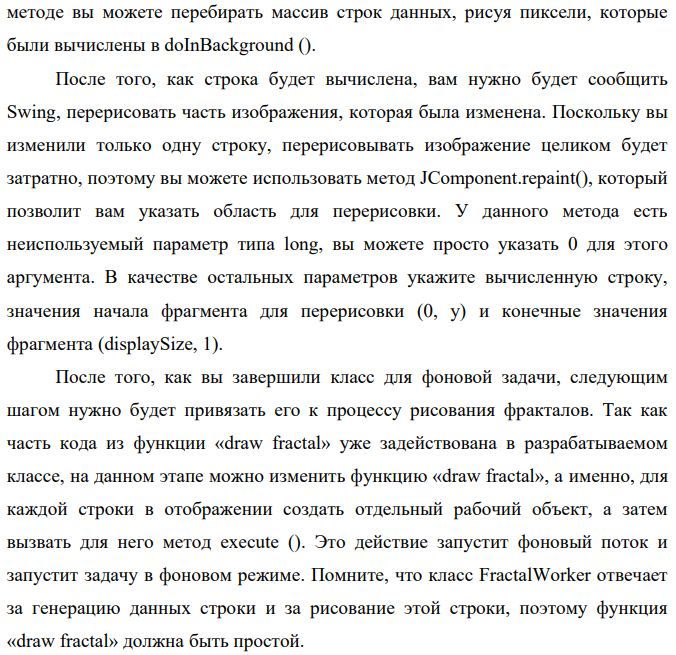


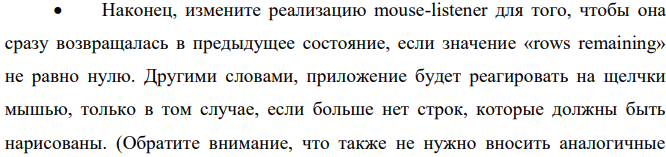
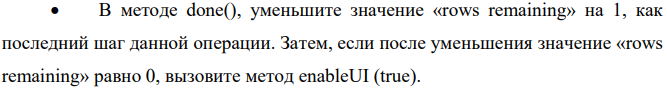
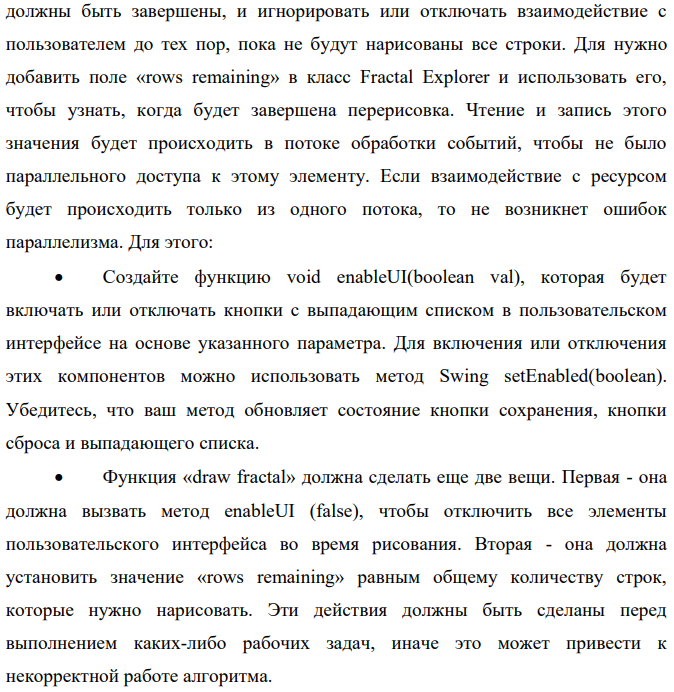
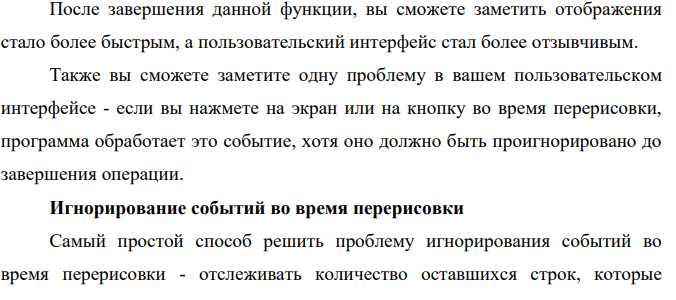


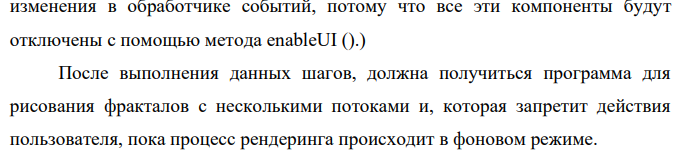












# Разработанные функции и классы

## 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

/\*\*

\* Класс отрисовывающий графический интерфейс Swing

\*/

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

/\*\*

\* Ширина и высота создаваемого окна

\*/

**private** **int** size;

/\*\*

\* Элемент управляющий изображением на форме

\*/

**private** JImageDisplay image;

/\*\*

\* Ссылка на генератор фракталов

\*/

**private** FractalGenerator fg;

/\*\*

\* Диапазон комплексной плоскости, выводимой на экран

\*/

**private** Rectangle2D.Double range;

/\*\* Конструктор, входные параметры:

\*

\* **@param** width - ширина создаваемого окна

\* **@param** height - длина создаваемого окна

\*/

/\*\*

\* Кнопка сброса

\*/

**private** JButton rbutton;

/\*\*

\* Кнопка сохранения

\*/

**private** JButton sbutton;

/\*\*

\* Выпадающий список факториалов

\*/

**private** JComboBox box;

/\*\*

\* Количество строк, которое необходимо обработать

\*/

**private** **int** rowsRemaining;

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

{

**this**.size = size;

image = **new** JImageDisplay(size, size);

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

fg = **new** Mandelbrot();

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

fg.getInitialRange(range);

sbutton = **new** JButton("Save Button");

box = **new** JComboBox();

}

/\*\*

\* Установка компонентов Swing, создание и показ формы

\*/

**private** **void** createAndShowGUI(){

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

**this**.setSize(size, size);

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

**this**.add(image);

rbutton.setActionCommand("reset");

rbutton.addActionListener(**new** ActionEventListener());

sbutton.setActionCommand("save");

sbutton.addActionListener(**new** ActionEventListener());

JPanel bpanel = **new** JPanel();

bpanel.add(rbutton);

bpanel.add(sbutton);

**this**.add(bpanel, "South");

image.addMouseListener(**new** MouseEventListener());

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

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

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

box.addActionListener(**new** ActionEventListener());

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

JPanel panel = **new** JPanel();

panel.add(label);

panel.add(box);

**this**.add(panel,"North");

// this.pack(); - После данного метода появляются отступы от картинки, почему не знаю

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

**this**.setResizable (**false**);

}

/\*\*

\* Метод вывода фрактала на экран

\*/

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

{

rowsRemaining = **this**.size;

enableUI(**false**);

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

{

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

row.execute();

}

}

/\*\*

\* Метод блокировки/разблокировки интерфейса

\* **@param** val

\*/

**void** enableUI(**boolean** val)

{

box.setEnabled(val);

sbutton.setEnabled(val);

rbutton.setEnabled(val);

}

/\*\* Обработчик событий\*\*/

**class** ActionEventListener **implements** ActionListener {

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

/\*

\* Обработка запускается в случае, если

\* действие вызвано объектом JComboBox

\*/

**if** (e.getSource() **instanceof** JComboBox)

{

JComboBox box = (JComboBox)e.getSource();

fg = (FractalGenerator)box.getSelectedItem();

fg.getInitialRange(range);

image.clearImage();

drawFractal();

}

/\*

\* Обработка запускается в случае, если действие

\* вызвано кнопкой с действием "save"

\*/

**else** **if**(((JButton)e.getSource()).getActionCommand() == "save")

{

// Получение объекта JFrame из кнопки количество getParent найдено методом тыка

JFrame jf = (JFrame)(((JButton)e.getSource()).getParent().getParent().getParent().getParent().getParent());

JFileChooser fc = **new** JFileChooser();

FileFilter filter = **new** FileNameExtensionFilter("PNG Images", "png");

fc.setFileFilter(filter);

fc.setAcceptAllFileFilterUsed(**false**);

**if**(fc.showSaveDialog(jf) != fc.***APPROVE\_OPTION***) System.***out***.println("Отмена сохранения");

**else**

{

File f = fc.getSelectedFile();

**try** {

ImageIO.*write*(image.bImage, "png", f);

} **catch** (IOException e1) {

JOptionPane.*showMessageDialog*(jf, e1.getMessage(), "Cannot Save Image", JOptionPane.***ERROR\_MESSAGE***);

}

}

}

/\*

\* Обработка зпускается в случае если действие

\* вызвано кнопкой не с действием "save"

\*/

**else**

{

fg.getInitialRange(range);

image.clearImage();

drawFractal();

}

}

}

/\*\* Обработчик нажатия на изображение \*\*/

**class** MouseEventListener **implements** MouseListener {

@Override

**public** **void** mouseClicked(MouseEvent e) {

**if** (rowsRemaining == 0)

{

**int** xCoord = e.getX();

**int** yCoord = e.getY();

**double** xCoor = FractalGenerator.*getCoord*(range.x, range.x + range.width,size, xCoord);

**double** yCoor = FractalGenerator.*getCoord*(range.y, range.y + range.width,size, yCoord);

fg.recenterAndZoomRange(range, xCoor, yCoor, 0.5);

drawFractal();

}

}

@Override

**public** **void** mouseEntered(MouseEvent e) {

// **TODO** Auto-generated method stub

}

@Override

**public** **void** mouseExited(MouseEvent e) {

// **TODO** Auto-generated method stub

}

@Override

**public** **void** mousePressed(MouseEvent e) {

// **TODO** Auto-generated method stub

}

@Override

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

// **TODO** Auto-generated method stub

}

}

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

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

app.createAndShowGUI();

app.drawFractal();

app.repaint();

}

/\*\*

\* Класс осуществляющий отрисовку строки изображения,

\* необходим для многопоточной обработки изображения

\*/

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

{

/\*\*

\* Значение координаты y, на которой вычисляются цвета строки

\*/

**private** **int** y;

/\*\*

\* Массив содержащий цвета для строки

\*/

**private** **int** rowRGB[];

/\*\*

\* Конструктор

\* **@param** y - координата для которой вычисляется строка

\*/

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

**this**.y = y;

}

/\*\*

\* Метод обработки в потоке, вычисляет массив цветов для строки

\*/

@Override

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

rowRGB = **new** **int**[size];

**for** (**int** x = 0;x<size;x++)

{

**double** xCoord = FractalGenerator.*getCoord* (range.x, range.x + range.width,size, x);

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

**int** iter = fg.numIterations(xCoord,yCoord);

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

**else**

{

// Выбор цвета в диапазоне от 0 до 16777216

// При возможных 2000 значениях

**int** color = 16777216/2000\*iter;

rowRGB[x] = color;

}

}

**return** **null**;

}

/\*\*

\* Метод вызываемый, при окончании обработки в потоке,

\* заполняет изображение вычисленными цветами и перересовывает его

\*/

@Override

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

{

**for** (**int** x = 0;x<size;x++)

{

image.drawPixel(x, y, rowRGB[x]);

}

image.repaint(0, 0, y, size, 1);

// Разблокировка интерфейса, по окончании обработки картинки

rowsRemaining -=1;

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

}

}

}

# Результат работы программы

