**Федеральное агентство связи**

Ордена Трудового Красного Знамени

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

Кафедра «Информатика»

Лабораторная работа №4: Рисование фракталов

по дисциплине «Кроссплатформенные технологии программирования»

Создать красивые изображения. Это будет сделано с помощью фреймворка Swing и Java API, который позволяет создавать графические пользовательские интерфейсы.

Выполнил: студент

группы БСТ1803

Шевцов Никита

Вариант №24

Москва 2020

**Содержание**

[1 Задание на разработку программы 4](#_Toc53411448)

[2 Разработанные функции и классы 9](#_Toc53411449)

[2.1 Класс FractalExplorer 9](#_Toc53411450)

[2.1.1 Метод void createAndShowGUI 9](#_Toc53411451)

[2.1.2 Метод void drawFractal 9](#_Toc53411452)

[2.1.3 Подкласс ActionListener button1\_Click с методом actionPerformed 9](#_Toc53411453)

[2.1.4 Подкласс MouseAdapter mouse1\_click с методом mouseReleased 9](#_Toc53411454)

[2.1.5 Метод main 9](#_Toc53411455)

[2.2. Класс JimageDisplay 10](#_Toc53411456)

[2.2.1 Метод void paintComponent 10](#_Toc53411457)

[2.2.2 Метод void clearImage 10](#_Toc53411458)

[2.2.3 Метод void drawPixel 10](#_Toc53411459)

[2.3 Класс Mandelbrot 10](#_Toc53411460)

[2.3.1 void getInitialRange 10](#_Toc53411461)

[2.3.2 int numIterations 11](#_Toc53411462)

[2.4 Класс Complex 11](#_Toc53411463)

[2.4.1 Метод Complex add 11](#_Toc53411464)

[2.4.2 Метод Complex sub 11](#_Toc53411465)

[2.4.3 Метод Complex mul 11](#_Toc53411466)

[2.4.4 Метод Complex div 11](#_Toc53411467)

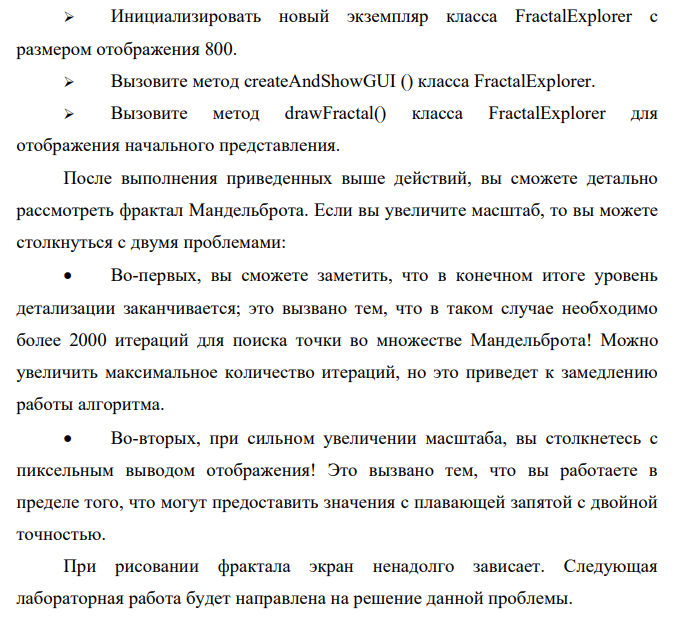
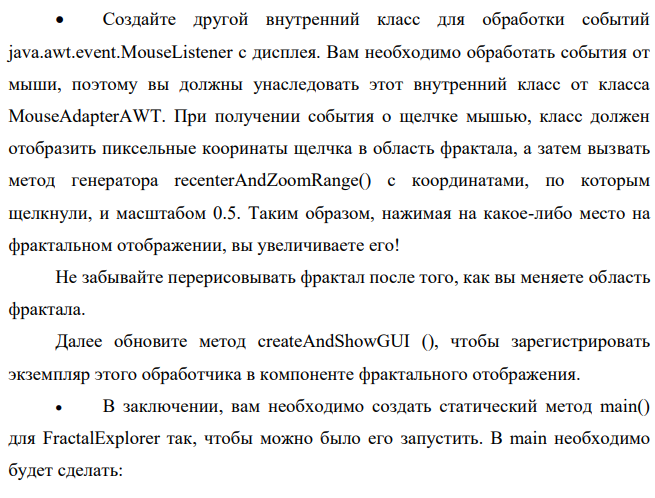
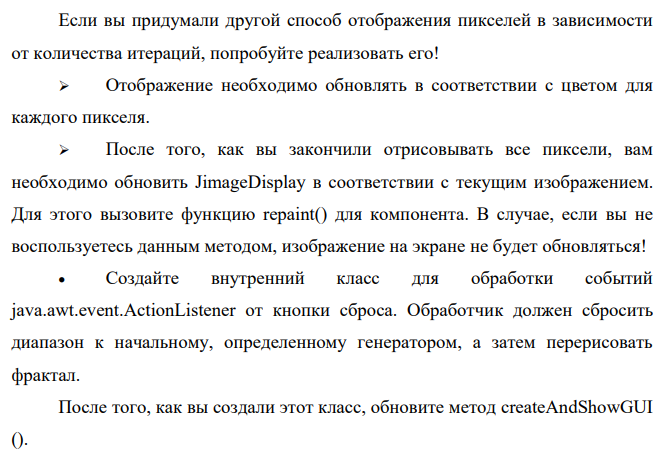
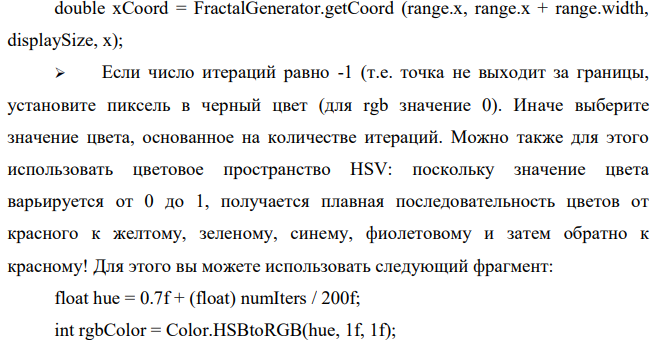
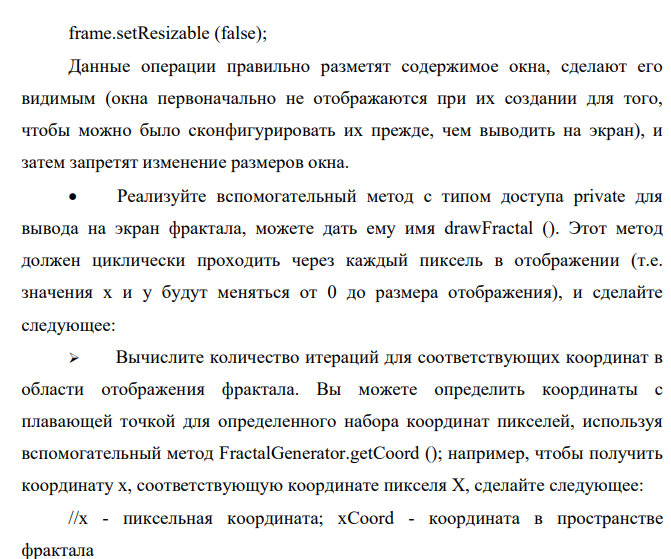
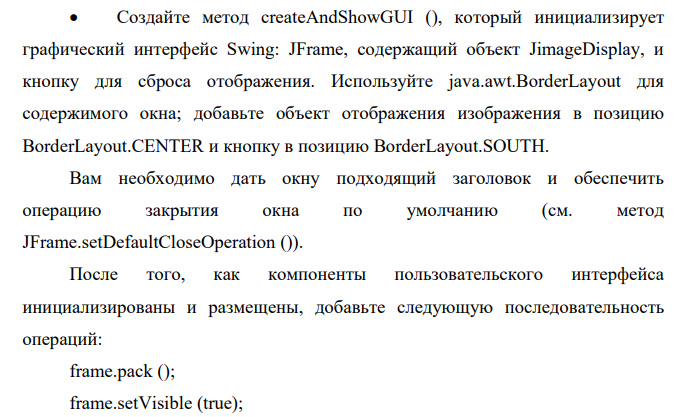
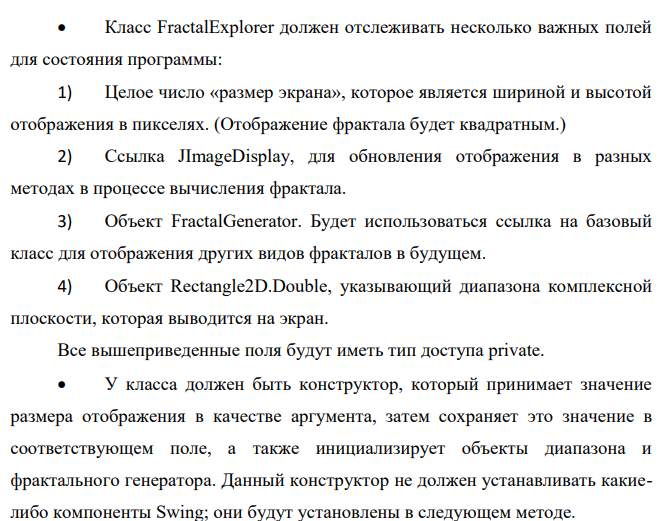
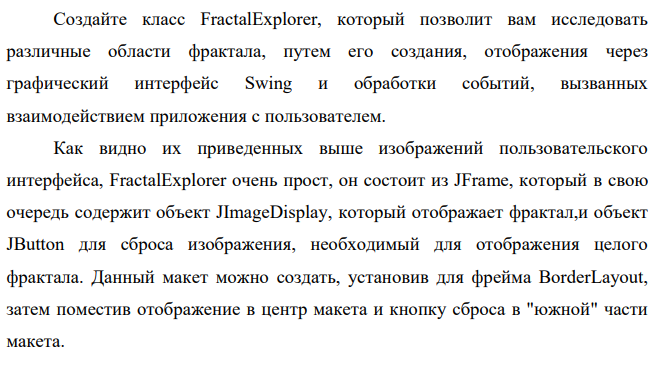
[2.4.5 Метод double absSqr 11](#_Toc53411468)

[2.4.6 Методы <T> <T>Value 12](#_Toc53411469)

[3 Код программы 12](#_Toc53411470)

[4 Результат работы программы 19](#_Toc53411471)

# Задание на разработку программы



# Разработанные функции и классы

## Класс FractalExplorer

Класс отрисовывающий графический интерфейс Swing.

* + 1. Метод void createAndShowGUI

Установка компонентов Swing, создание и показ формы

* + 1. Метод void drawFractal

Метод вывода фрактала на экран

* + 1. Подкласс ActionListener button1\_Click с методом actionPerformed

Обработчик нажатия на кнопку

* + 1. Подкласс MouseAdapter mouse1\_click с методом mouseReleased

Обработчик нажатия на изображение

* + 1. Метод main

Создает объект формы, создает на ней и отрисовывает объекты Swing и фрактал

## 2.2. Класс JimageDisplay

Класс для управления изображением на форме Swing

* + 1. Метод void paintComponent

Перерисовывает изображение на экране

* + 1. Метод void clearImage

Метод, очищающий изображение, устанавливая все пиксели в черный цвет

* + 1. Метод void drawPixel

Метод устанавливающий пиксель в определённый цвет

## Класс Mandelbrot

Реализует методы, необходимые для генерации фрактала Мандельброта

* + 1. void getInitialRange

Метод определяющий область комплексной плоскости для фрактала Мандельброта

* + 1. int numIterations

Метод, вычисляющий количество итераций для отрисовки точки

## Класс Complex

Класс описывающий комплексную переменную

* + 1. Метод Complex add

Метод сложения комплексных чисел

* + 1. Метод Complex sub

Метод вычитания комплексных чисел

* + 1. Метод Complex mul

Метод умножения комплексных чисел

* + 1. Метод Complex div

Метод деления комплексных чисел

* + 1. Метод double absSqr

Метод, возвращающий квадрат модуля комплексного числа

* + 1. Методы <T> <T>Value

Методы, возвращающие модуль комплексного числа в выбраном типе

# Код программы

/\*\*

\* Класс комплексных чисел

\* **@author** niksh

\* **@version** 1.0

\*/

**public** **class** Complex **extends** Number{

**private** **static** **final** **long** ***serialVersionUID*** = 7746583188569394990L;

/\*\* Реальная составляющая числа \*/

**private** **double** real;

/\*\* Комплексная составляющая числа \*/

**private** **double** imag;

/\*\*

\* Конструктор - создает комплексное число с заданными значениями

\* **@param** real - реальная составляющая числа

\* **@param** imag - комплексная составляющая числа

\*/

**public** Complex(**double** real, **double** imag) {

**this**.real = real;

**this**.imag = imag;

}

/\*\*

\* Конструктор - создает комплексное число 0 + i0

\*/

**public** Complex() {

**this**(0,0);

}

/\*\*

\* Операция сложения комплексных чисел

\* **@param** n - комплексное число для сложения

\* **@return** Комплексная сумма

\*/

**public** Complex add(Complex n) {

Complex res = **new** Complex();

res.real = **this**.real + n.real;

res.imag = **this**.imag + n.imag;

**return** res;

}

/\*\*

\* Операция сложения комплексного и обычного числа

\* **@param** n - число для сложения

\* **@return** Комплексная сумма

\*/

**public** Complex add(Number n) {

Complex res = **new** Complex(**this**.real, **this**.imag);

res.real += (**double**) n;

**return** res;

}

/\*\*

\* Операция вычитания комплексных чисел

\* **@param** n - комплексное число для вычитания

\* **@return** Комплексная разность

\*/

**public** Complex sub(Complex n) {

Complex res = **new** Complex();

res.real = **this**.real - n.real;

res.imag = **this**.imag - n.imag;

**return** res;

}

/\*\*

\* Операция вычитания комплексного и обычного числа

\* **@param** n - число для вычитания

\* **@return** Комплексная разность

\*/

**public** Complex sub(Number n) {

Complex res = **new** Complex(**this**.real, **this**.imag);

res.real -= (**double**) n;

**return** res;

}

/\*\*

\* Операция умножения комплексных чисел

\* **@param** n - комплексное число для умножения

\* **@return** Комплексное произведение

\*/

**public** Complex mul(Complex n) {

Complex res = **new** Complex();

res.real = **this**.real\*n.real - **this**.imag\*n.imag;

res.imag = **this**.real\*n.imag + **this**.imag\*n.real;

**return** res;

}

/\*\*

\* Операция умножения комплексного и обычного числа

\* **@param** n - число для умножения

\* **@return** Комплексное произведение

\*/

**public** Complex mul(Number n) {

Complex res = **new** Complex(**this**.real, **this**.imag);

res.real \*= (**double**)n;

res.imag \*= (**double**)n;

**return** res;

}

/\*\*

\* Операция деления комплексных чисел

\* **@param** n - комплексное число для деления

\* **@return** Комплексное частное

\*/

**public** Complex div(Complex n) {

Complex res = **new** Complex();

**double** divisor = n.real\*n.real + n.imag\*n.imag;

res.real = (**this**.real\*n.real + **this**.imag\*n.imag)/divisor;

res.imag = (**this**.imag\*n.real - **this**.real\*n.imag)/divisor;

**return** res;

}

/\*\*

\* Операция деления комплексного и обычного числа

\* **@param** n - число для деления

\* **@return** Комплексное частное

\*/

**public** Complex div(Number n) {

Complex res = **new** Complex(**this**.real, **this**.imag);

res.real /= (**double**) n;

res.imag /= (**double**) n;

**return** res;

}

/\*\*

\* Квадрат модуля комплексного числа

\*/

**public** **double** absSqr() {

**return** **this**.real\***this**.real + **this**.imag\***this**.imag;

}

/\*\* Модуль комплексного числа типа <code>double</code> \*/

@Override

**public** **double** doubleValue() {

**double** res = Math.*sqrt*(absSqr());

**return** res;

}

/\*\* Модуль комплексного числа типа <code>float</code> \*/

@Override

**public** **float** floatValue() {

**float** res = (**float**) Math.*sqrt*(absSqr());

**return** res;

}

/\*\* Модуль комплексного числа типа <code>int</code> \*/

@Override

**public** **int** intValue() {

**int** res = (**int**) Math.*sqrt*(absSqr());

**return** res;

}

/\*\* Модуль комплексного числа типа <code>long</code> \*/

@Override

**public** **long** longValue() {

**long** res = (**long**) Math.*sqrt*(absSqr());

**return** res;

}

}

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

/\*\*

\* Класс, управляющий расчетом и начальным отображением фрактала Мандельброта

\* **@author** niksh

\*

\*/

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

/\*\* Максимальное кол-во итераций при расчете \*/

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

/\*\* Размеры отображаемого участка (DIM\*DIM) \*/

**public** **static** **final** **double** ***REC\_DIMENSION*** = 3;

/\*\* x центр отображаемого участка \*/

**public** **static** **final** **double** ***REC\_CENTER\_X*** = -2;

/\*\* y центр отображаемого участка \*/

**public** **static** **final** **double** ***REC\_CENTER\_Y*** = -1.5;

/\*\*

\* Установка начального диапазона отображения фрактала

\*/

@Override

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

range.height = ***REC\_DIMENSION***;

range.width = ***REC\_DIMENSION***;

range.x = ***REC\_CENTER\_X***;

range.y = ***REC\_CENTER\_Y***;

}

/\*\*

\* Итеративная формула расчета принадлежности точки к множеству Мандельброта

\* **@param** x - x координата, приведенная к реальной части числа на комплексной плоскости

\* **@param** y - y координата, приведенная к комплексной части числа на комплексной плоскости

\* **@return** Кол-во итераций расчета до выхода за пределы множества, -1 если расчеты не вышли за пределы

\*/

@Override

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

Complex coord = **new** Complex(x,y);

Complex zIter = **new** Complex();

**int** iter = 0;

zIter = zIter.add(coord);

**while** (zIter.absSqr() < 4 & iter < ***MAX\_ITERATIONS***) {

iter += 1;

zIter = zIter.mul(zIter).add(coord);

}

**if** (iter == 2000) **return** -1;

**return** iter;

}

}

import javax.swing.JComponent;

import java.awt.image.\*;

import java.awt.\*;

/\*\*

\* Класс элемента, управляющий отображением изображения

\* @author niksh

\*/

public class JImageDisplay extends JComponent {

private static final long serialVersionUID = -611291715804825948L;

/\*\* Ширина элемента в пикселях \*/

private int width;

/\*\* Высота элемента в пикселях \*/

private int height;

/\*\* Объект изображения \*/

private BufferedImage image;

/\*\* Стандартный размер окна \*/

public static final int DEFAULT\_SIZE = 500;

/\*\*

\* Конструктор - создает элемент интерфейса, содержащий изображение

\* @param width - ширина элемента

\* @param height - высота элемента

\*/

public JImageDisplay(int width, int height) {

this.width = width;

this.height = height;

this.image = new BufferedImage(width, height, BufferedImage.TYPE\_INT\_RGB); // Создать изображение width\*height с цветовым пространством RGB

super.setPreferredSize(new Dimension(width,height)); // Задать размер элемента равным размеру изображения

}

/\*\*

\* Конструктор - создает элемент интерфейса, размером 500 на 500, содержащий изображение

\*/

public JImageDisplay() {

this(DEFAULT\_SIZE,DEFAULT\_SIZE);

}

/\*\*

\* Переопределение метода paintComponent, который отображает наш элемент

\*/

@Override

protected void paintComponent(Graphics g) {

g.drawImage (image, 0, 0, this.width, this.height, null);

}

/\*\*

\* Метод, устанавливающий все пиксели изображения в черный цвет

\*/

public void clearImage() {

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

for (int x = 0; x < this.width; x++) {

this.image.setRGB(x, y, 0);

}

}

}

/\*\*

\* Метод, устанавливающий выбранный пиксель изображения в указанный цвет

\* @param x - x координата пикселя (ширина)

\* @param y - y координата пикселя (высота)

\* @param rgbColor - код цвета

\* @see Color#getRGB()

\*/

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

this.image.setRGB(x, y, rgbColor);

}

}

import java.awt.\*;

import java.awt.event.\*;

import javax.swing.\*;

import java.awt.geom.Rectangle2D;

/\*\*

\* Приложение для отображения и исследования фракталов

\* @author niksh

\* @version 1.0

\*/

public class FractalExplorer extends JFrame {

private static final long serialVersionUID = -8728543713519701421L;

/\*\* 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;

/\*\*

\* Конструктор - инициализирует компоненты GUI и отображения фрактала

\* @param size - размер окна

\*/

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);

}

public FractalExplorer() {

this(JImageDisplay.DEFAULT\_SIZE);

}

public void createAndShowGUI() {

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

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

JButton button = new JButton("Сброс");

button.addActionListener(new button1\_Click());

this.add(button, BorderLayout.SOUTH);

this.setTitle("Fractal Explorer");

this.setDefaultCloseOperation(EXIT\_ON\_CLOSE);

this.pack();

this.setVisible(true);

this.setResizable(true);

}

private void drawFractal() {

int iter;

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

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

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

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

iter = fgen.numIterations(real, imag);

if (iter == -1) image.drawPixel(x, y, 0);

else {

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

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

image.drawPixel(x, y, rgbColor);

}

}

}

image.repaint();

}

private class button1\_Click implements ActionListener {

@Override

public void actionPerformed(ActionEvent arg0) {

fgen.getInitialRange(range);

drawFractal();

}

}

private class mouse1\_Click extends MouseAdapter {

@Override

public void mouseReleased(MouseEvent arg0) {

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(1000);

app.createAndShowGUI();

app.drawFractal();

}

}

# Результат работы программы

