**Федеральное государственное общеобразовательное учреждение высшего образования**

**Московский государственный технический университет**

**им. Н. Э. Баумана**

Факультет "Информатика и системы управления"

Кафедра "Информационные системы и телекоммуникации"

**Отчет по лабораторной работе №4**

**"Программа для изучения фракталов"**

Студент: Задубин А.А.

Группа: ИУ3-42Б

Преподаватель: Руденкова Ю.С.

**Задание:**

Создание пользовательского интерфейса.

Создали класс JImageDisplay наследованный от javax.swing.JComponent. Класс должен был иметь одно private поле, экземпляр класса java.awt.image.BufferedImage. Конструктор JImageDisplay получает целые ширину и высоту и инициализировать поле BufferedImage ссылкой на новое изображение заданной ширины и высоты. Тип изображения равен TYPE\_INT\_RGB.

Конструктор вызывает метод родительского класса setPreferredSize() передавая ему заданные ширину и высоту. Тогда компонент, который создали, начинает прорисовывать изображение после того как будет добавлен к пользовательскому интерфейсу.

Пользовательский компонент Swing сам прорисовывает себя переопределив метод protected paintComponent(Graphics g) класса JComponent. Реализация paintComponent(g) суперкласса вызывается в любом случае, для того чтобы все рамки и другие элементы правильно прорисовывались. После вызова метода суперкласса, мы можем нарисовать изображение в области компонента, используя подобные операции: g.drawImage(image, 0, 0, image.getWidth(), image.getHeight(), null); (Мы указываем null в аргументе ImageObserver, потому что не нуждаемся в этом функционале).

Нам также потребовалось создать два public метода для записи данных в изображение: clearImage() который устанавливает черный цвет для всех пикселов изображения (значение RGB 0) и drawPixel(int x, int y, int rgbColor) которых задает указанный цвет пиксела. Оба эти метода использовуют один из методов setRGB() класса BufferedImage.

Вычисление фрактала Мандельброта.

Создали дочерний класс FractalGenerator с именем Mandelbrot. Реализовали только два метода, getInitialRange() и numIterations().

Метод getInitialRange(Rectangle2D.Double) позволяет указать генератору фрактала, какая часть комплектной области "интересна" для вычисления фрактала. Объект прямоугольник передается как аргумент метода, и метод изменяет поля прямоугольника так, чтобы он содержал правильный начальный диапазон вычислений фрактала.

Собираем все вместе.

Создали класс FractalExplorer, который позволяет изучить различные фрагменты фрактала, вычисляя их и отображая на экране с помощью графического интерфейса Swing, а также обработать различные сообщения от пользователя.

В классе FractalExplorer храним некоторую важную информацию состоянии программы: o Целое значение "размера экрана", число, задающее одновременно высоту и ширину области отображения. Ссылка на объект JImageDisplay, для того чтобы мы имели возможность изменять изображение из различных методов класса по мере вычисления фрактала. Объект FractalGenerator. Здесь использовали ссылку на базовый класс, для того чтобы в будущих версиях можно было показывать на экране различные типы фракталов. Объект Rectangle2D.Double, который задает отображаемый диапазон в комплексной области.

В конце, после того как компоненты пользовательcкого интерфейса проинициализированы и размещены в окне, добавили эту последовательность вызовов: frame.pack(); frame.setVisible(true); frame.setResizable(false);

**Код программы:**

FractalExplorer.java

**import** **java.awt.BorderLayout**;

**import** **java.awt.Color**;

**import** **java.awt.event.ActionEvent**;

**import** **java.awt.event.ActionListener**;

**import** **java.awt.event.MouseAdapter**;

**import** **java.awt.event.MouseEvent**;

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

**import** **javax.swing.JButton**;

**import** **javax.swing.JFrame**;

**public** **class** **FractalExplorer** {

**private** **int** side;

**private** JImageDisplay jpgImg;

**private** FractalGenerator geni = **new** Mandelbrot();

**private** Rectangle2D.Double vsblRange;

**public** **FractalExplorer**(**int** x) {

side = x;

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

geni.getInitialRange(vsblRange);

jpgImg = **new** JImageDisplay(x, x);

}

**class** **ActiveLisner** **implements** ActionListener {

**@Override**

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

System.out.println(e.getActionCommand());

**if** (e.getActionCommand().equals("Сброс")) {

System.out.println("Сбрасывание");

jpgImg.clearImage();

geni.getInitialRange(vsblRange);

drawFractal();

}

}

}

**class** **MouseClass** **extends** MouseAdapter {

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

System.out.println(e);

**double** xCoord = FractalGenerator.getCoord(vsblRange.x, vsblRange.x + vsblRange.width, side, e.getX());

**double** yCoord = FractalGenerator.getCoord(vsblRange.y, vsblRange.y + vsblRange.height, side, e.getY());

geni.recenterAndZoomRange(vsblRange, xCoord, yCoord, **0.3**);

drawFractal();

}

}

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

**int** a;

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

**for** (**int** j = **0**; j < side; j++) {

**double** xCoord = FractalGenerator.getCoord(vsblRange.x, vsblRange.x + vsblRange.width, side, i);

**double** yCoord = FractalGenerator.getCoord(vsblRange.y, vsblRange.y + vsblRange.height, side, j);

a = geni.numIterations(xCoord, yCoord);

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

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

**if** (a == -**1**) {

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

} **else** {

jpgImg.drawPixel(i, j, rgbColor);

}

}

}

jpgImg.repaint();

}

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

JFrame f = **new** JFrame("Лабораторная работа 4");

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

f.add(jpgImg, BorderLayout.CENTER);

JButton b = **new** JButton("Сброс");

f.add(b, BorderLayout.SOUTH);

b.addActionListener(**new** ActiveLisner());

jpgImg.addMouseListener(**new** MouseClass());

f.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

f.pack();

f.setVisible(**true**);

f.setResizable(**false**);

}

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

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

frac.createAndShowGUI();

frac.drawFractal();

}

}

FractalGenerator.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);

}

JImageDisplay.java

**import** **java.awt.Dimension**;

**import** **java.awt.Graphics**;

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

**import** **javax.swing.JComponent**;

**public** **class** **JImageDisplay** **extends** JComponent {

**private** BufferedImage img;

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

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

setPreferredSize(**new** Dimension(width, height));

}

**@Override**

**protected** **void** **paintComponent**(Graphics g) {

g.drawImage(img, **0**, **0**, img.getWidth(), img.getHeight(), **null**);

}

**public** **void** **clearImage**() {

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

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

img.setRGB(i, j, **0**);

}

}

}

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

img.setRGB(x, y, rgbColor);

}

}

Mandelbrot.java

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

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

**public** **static** **final** **int** MAX\_ITERATIONS = **1000**;

**@Override**

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

range.x = -**2**;

range.y = -**1.5**;

range.width = **3**;

range.height = **3**;

}

**@Override**

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

**double** xn = **0**, yn = **0**;

**double** zk = **0**, tmp = **0**;

**int** i;

**for** (i = **0**; i < MAX\_ITERATIONS; i++) {

tmp = xn \* xn - yn \* yn + x;

yn = **2** \* xn \* yn + y;

xn = tmp;

zk = xn \* xn + yn \* yn;

**if** (zk > **4**) **return** i;

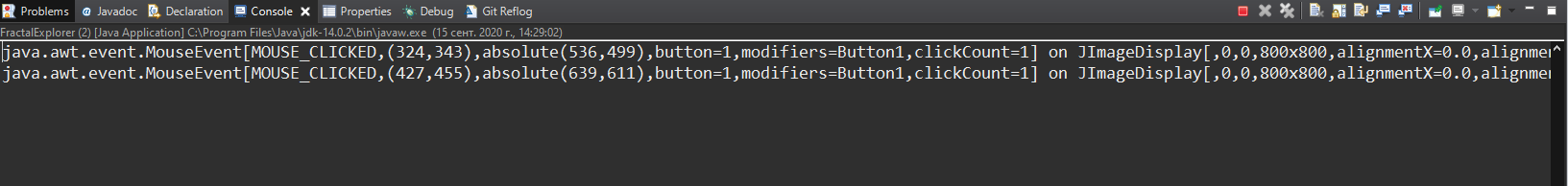
}

**return** (i == MAX\_ITERATIONS) ? -**1** : i;

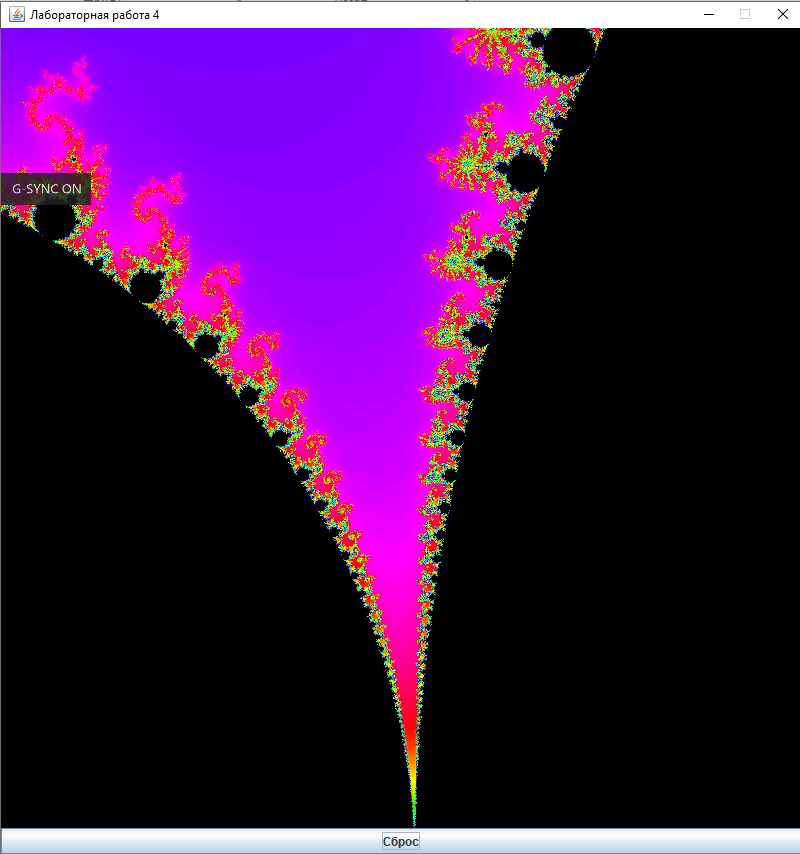
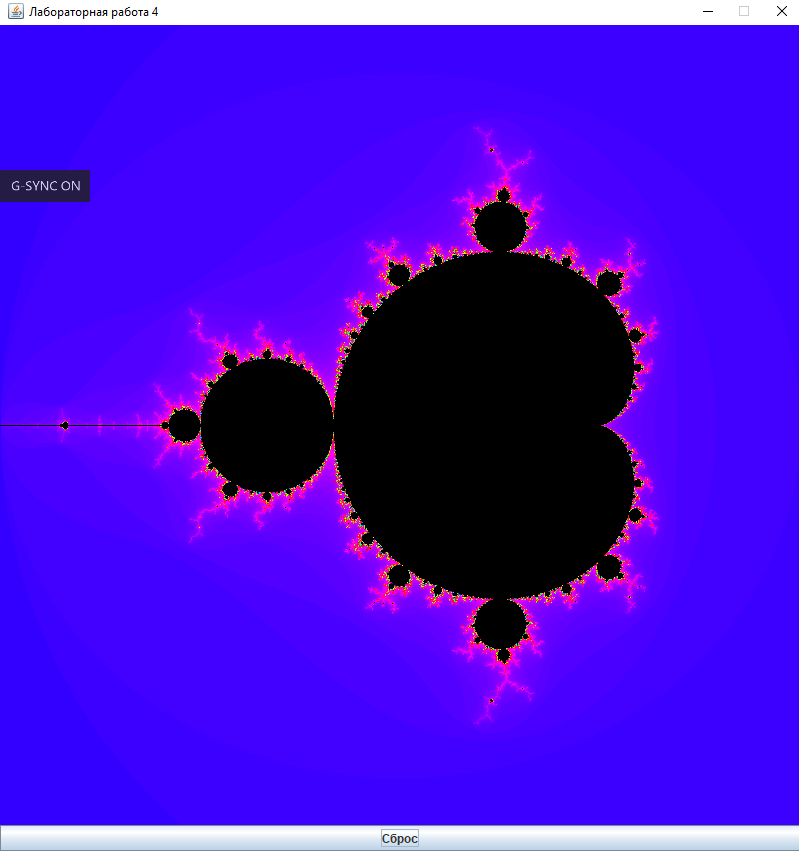
}

}

**Вывод в консоль:**

****

**Результат работы программы:**



**Вывод:** Мы изучили фрактал Мандельброта. Во-первых, мы обнаружили что уровень детализации в какой-то момент перестает увеличиваться; это происходит потому что нам требуется более 2000 итераций для того чтобы определить принадлежит точка множеству Мандельброта или нет. Конечно, мы можем увеличить максимальное число итераций, но тогда черные области фрактала уменьшат скорость работы нашей программы. Во-вторых, если мы сделаем увеличение очень большим, мы, в конце концов, увидим увеличенные пикселы. Это происходит потому, что достигнут предел точности представления чисел с плавающей точкой в формате double.