**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 №6**

**"Многозадачность в программе FractalExplorer"**

Студент: Задубин А.А.

Группа: ИУ3-42Б

Преподаватель: Руденкова Ю.С.

Задание:

Надо было создать дочерний класс SwingWorker и дать ему имя FractalWorker. Это класс должен был быть внутренним классом FractalExplorer - private class FractalWorker extends SwingWorker.

Класс FractalWorker отвечает за вычисление цветов точек одной строки фрактала, и для этого ему потребовалось два поля: целая координата y вычисляемой строки, и массив целых чисел (int) для хранения вычисленных RGB значений каждой точки в строке. Конструктор через аргумент получает координату y и сохранять ее.

Метод done() вызывается когда фоновая задача завершена, и этот метод вызывается в контексте потока диспетчера событий Swing.

После завершения прорисовки строки, надо было попросить Swing перерисовать часть экрана, которая была изменена. Так как мы изменили только одну строку воспользуемся вариантом JComponent.repaint() который позволяет задать границы области перерисовки. Метод имеет небольшую странность – у него есть неиспользуемый параметр типа long в начале списка аргументов, и мы просто подставляем ему значение 0. В остальном, просто указываем строку, которую надо перерисовать – позицию (0, y) и размер (displaySize, 1).

Создали функцию void enableUI(boolean val) которая блокирует или разрешает работу кнопок и выпадающего списка в вашем пользовательском интерфейсе в зависимости от значения аргумента. Используем метод setEnabled(boolean) Swing компонента для блокировки/разрешения работы. Необходимо менять состояние кнопки сохраняющей изображение, кнопки сбрасывающей изображение в начальное состояние и выпадающего списка.

В методе done() фонового потока, уменьшаем "количество отавшийся строк" на единицу как завершающий шаг операции. Затем, если количество оставшихся строк становится равным 0, вызываем enableUI(true).

**Код программы:**

BurningShip.java

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

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

**public** **static** **final** **int** MAX\_ITERATIONS = **500**;

**@Override**

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

range.x = -**2**;

range.y = -**2.5**;

range.width = **4**;

range.height = **4**;

}

**@Override**

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

// xn = Re, yn = lm;

**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** \* Math.abs(xn) \* Math.abs(yn) + y;

xn = tmp;

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

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

}

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

}

**@Override**

**public** String **toString**() {

**return** "Горящий корабль";

}

}

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** **java.awt.image.RenderedImage**;

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

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

**import** **javax.swing.JComboBox**;

**import** **javax.swing.JFileChooser**;

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

**import** **javax.swing.JLabel**;

**import** **javax.swing.JOptionPane**;

**import** **javax.swing.JPanel**;

**import** **javax.swing.SwingWorker**;

**import** **javax.swing.filechooser.FileFilter**;

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

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

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

**private** **int** strCount;

**private** JButton b;

**private** JButton b2;

**private** JComboBox<FractalGenerator> combo;

**private** JImageDisplay jpgImg;

// private FractalGenerator geni = new Mandelbrot();

// private FractalGenerator geni = new Tricorn();

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

**private** Rectangle2D.Double vsblRange;

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

side = x;

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

geni.getInitialRange(vsblRange);

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

}

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

b.setEnabled(val);

b2.setEnabled(val);

combo.setEnabled(val);

}

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

**@Override**

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

String cmd = e.getActionCommand();

**if** (cmd.equals("Ресет")) {

jpgImg.clearImage();

geni.getInitialRange(vsblRange);

drawFractal();

} **else** **if** (cmd.equals("comboBoxChanged")) {

JComboBox<FractalGenerator> combo = (JComboBox<FractalGenerator>) e.getSource();

geni = (FractalGenerator) combo.getSelectedItem();

geni.getInitialRange(vsblRange);

drawFractal();

} **else** **if** (cmd.equals("Скрин")) {

JFileChooser choozer = **new** JFileChooser();

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

choozer.setFileFilter(filter);

choozer.setAcceptAllFileFilterUsed(**false**);

**int** a = choozer.showSaveDialog(choozer.getParent());

System.out.println(e);

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

String png = "png";

**try** {

ImageIO.write((RenderedImage) jpgImg.getImg(), png, choozer.getSelectedFile());

} **catch** (Exception ex) {

JOptionPane.showMessageDialog(choozer.getParent(), ex.getMessage(), "Ошибка сохранения",

JOptionPane.ERROR\_MESSAGE);

}

}

}

}

}

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

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

**if** (strCount == **0**) {

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**() {

enableUI(**false**);

strCount = side;

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

**new** **FractalWorker**(i).execute();

}

}

////

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

JFrame f = **new** JFrame("Лабораторная работа 6");

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

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

f.add(jpgImg, BorderLayout.CENTER);

b = **new** JButton("Ресет");

JPanel p2 = **new** JPanel();

b2 = **new** JButton("Скрин");

f.add(b, BorderLayout.SOUTH);

p2.add(b2);

p2.add(b);

f.add(p2, BorderLayout.SOUTH);

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

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

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

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

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

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

JPanel p = **new** JPanel();

p.add(**new** JLabel("Фрактал "));

p.add(combo);

f.add(p, BorderLayout.NORTH);

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

b2.addActionListener(**new** ActiveLisner());

f.pack();

f.setVisible(**true**);

f.setResizable(**false**);

}

// Потоки

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

**int** y;

**int**[] yarray;

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

**this**.y = y;

}

**@Override**

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

yarray = **new** **int**[side];

**int** a;

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

**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);

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**) {

yarray[i] = **0**;

} **else** {

yarray[i] = rgbColor;

}

// }

}

**return** **null**;

}

**@Override**

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

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

jpgImg.drawPixel(i, y, yarray[i]);

jpgImg.repaint(**0**, **0**, i, side, **1**);

}

strCount--;

**if** (strCount == **0**)

enableUI(**true**);

}

}

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

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

frac.createAndShowGUI();

frac.drawFractal();

}

}

FractalGenerator.java

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

/\*\*

\* This class provides the common interface and operations for fractal

\* generators that can be viewed in the Fractal Explorer.

\*/

**public** **abstract** **class** **FractalGenerator** {

/\*\*

\* This static helper function takes an integer coordinate and converts it

\* into a double-precision value corresponding to a specific range. It is

\* used to convert pixel coordinates into double-precision values for

\* computing fractals, etc.

\*

\* @param rangeMin the minimum value of the floating-point range

\* @param rangeMax the maximum value of the floating-point range

\*

\* @param size the size of the dimension that the pixel coordinate is from.

\* For example, this might be the image width, or the image height.

\*

\* @param coord the coordinate to compute the double-precision value for.

\* The coordinate should fall in the range [0, size].

\*/

**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);

}

/\*\*

\* Sets the specified rectangle to contain the initial range suitable for

\* the fractal being generated.

\*/

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

/\*\*

\* Updates the current range to be centered at the specified coordinates,

\* and to be zoomed in or out by the specified scaling factor.

\*/

**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;

}

/\*\*

\* Given a coordinate <em>x</em> + <em>iy</em> in the complex plane,

\* computes and returns the number of iterations before the fractal

\* function escapes the bounding area for that point. A point that

\* doesn't escape before the iteration limit is reached is indicated

\* with a result of -1.

\*/

**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));

}

**public** BufferedImage **getImg**() {

**return** img;

}

**@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;

}

**@Override**

**public** String **toString**() {

**return** "Мандельброта";

}

}

Tricorn.java

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

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

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

**@Override**

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

range.x = -**2**;

range.y = -**2**;

range.width = **4**;

range.height = **4**;

}

**@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;

}

**@Override**

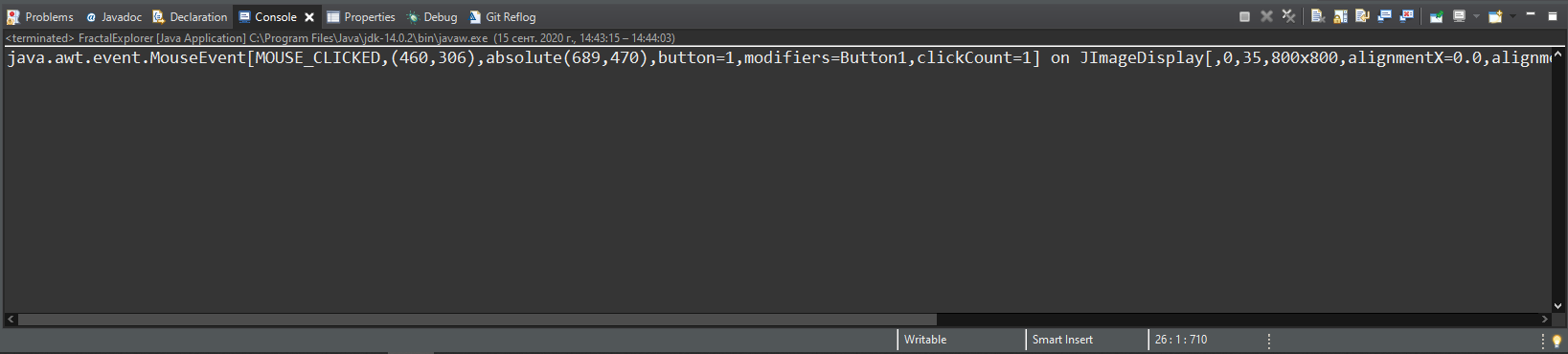
**public** String **toString**() {

**return** "Треуголка";

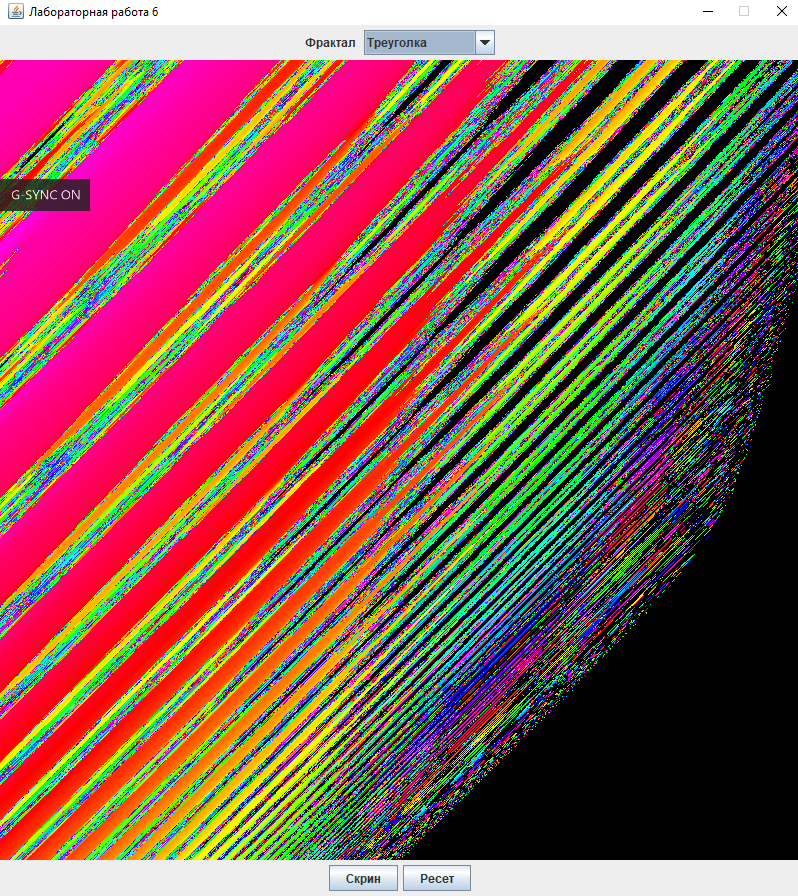
}

}

**Вывод в консоль:**

****

**Результат работы программы:**



**Вывод**: мы модифицировали старую программу, ускорив ее работу и запретив отклик на события во время прорисовки фрактала.