**Федеральное агентство связи**

**Ордена Трудового Красного Знамени**

**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Московский технический университет связи и информатики»**

**Кафедра Информатики**



**Отчет по лабораторной работе №6**

по предмету «КТП»

Выполнил: студент группы БВТ1802

Ткачев Александр Владимирович

Руководитель:

Ксения Андреевна Полянцева

Москва 2020

**1 Цель работы**

Цель работы: изучить алгоритм расчета фрактала, научиться исользовать java.swing и класс java.swing.worker.

**2 Задание**

Распараллелить отрисовку фрактала, сделав ее более быстрой.

**3 Текст программы**

**Class FractalExplorer**

package com.company;  
  
import java.awt.\*;  
import javax.swing.\*;  
import java.awt.geom.Rectangle2D;  
import java.awt.event.\*;  
import javax.swing.JFileChooser.\*;  
import javax.swing.filechooser.\*;  
import javax.imageio.ImageIO.\*;  
import java.awt.image.\*;  
  
*/\*\*  
 \* Этот класс позволяет исследовать различные части фрактала благодаря  
 \* созданию и отображению Swing GUI и обрабатвывает события вызванные  
 \* пользователем.  
 \*/*public class FractalExplorer  
{  
 */\*\* Поля для кнопки "save", "reset" и списка для enableUI. \*\*/* private JButton saveButton;  
 private JButton resetButton;  
 private JComboBox myComboBox;  
  
 */\*\* Количество строк, оставшиеся для рисования. \*\*/* private int rowsRemaining;  
  
 */\*\* Целочисленный размер экрана. Длина и высота экарана в пикселях. \*/* private int displaySize;  
  
 */\*\*  
 \* JImageDisplay ссылка на обновление отображения из различных методов  
 \* при вычисленнии фрактала.  
 \*/* private JImageDisplay display;  
  
 */\*\*  
 \* FractalGenerator объект использует ссылку на базовый класс, чтобы показать другие  
 \* типы фракталов в будущем.  
 \*/* private FractalGenerator fractal;  
  
 */\*\*  
 \* Rectangle2D.Double объект, который определяет диапозон комплекса,  
 \* который отображается.  
 \*/* private Rectangle2D.Double range;  
  
 */\*\*  
 \* Конструктор, который принимает размер экрана, сохраняет его и  
 \* устанавливает объекты диапозона и фрактального генератора.  
 \*/* public FractalExplorer(int size) {  
 */\*\* Размер экрана. \*/* displaySize = size;  
  
 */\*\* Устанавливает объекты фрактального генератора и диапозона. \*/* fractal = new Mandelbrot();  
 range = new Rectangle2D.Double();  
 fractal.getInitialRange(range);  
 display = new JImageDisplay(displaySize, displaySize);  
  
 }  
  
 */\*\*  
 \* Устанавливает Swing GUI с помощью JFrame, содержащего объект  
 \* JImageDisplay и кнопку для сброса экрана, кнопку для сохранения  
 \* текущего фрактала, JComboBox для выбора типа фрактала.  
 \* JComboBox содержит панель с именами.  
 \*/* public void createAndShowGUI()  
 {  
 */\*\*  
 \* Устанавливает фрэйм, чтобы использовать java.awt.BorderLayout,  
 \* для его содержимого.  
 \*/* display.setLayout(new BorderLayout());  
 JFrame myFrame = new JFrame("Fractal Explorer");  
  
 */\*\*  
 \* Добавляет объект экранного изображения в поизицию  
 \* BorderLayout.CENTER.  
 \*/* myFrame.add(display, BorderLayout.*CENTER*);  
  
 */\*\* Создание кнопки сброса. \*/* resetButton = new JButton("Reset");  
  
 */\*\* Экземпляр ButtonHandler для кнопки reset. \*/* ButtonHandler resetHandler = new ButtonHandler();  
 resetButton.addActionListener(resetHandler);  
  
 */\*\* Экземпляр для MouseHandler компонента фрактал-дисплея. \*/* MouseHandler click = new MouseHandler();  
 display.addMouseListener(click);  
  
 */\*\*  
 \* Устанавливает значение "exit" по умолчанию для операции  
 \* закрытия кадра.  
 \*/* myFrame.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
  
 */\*\* Устанавливает поле со списком. \*/* myComboBox = new JComboBox();  
  
 */\*\* Добавляет каждый объект фрактального типа в поле со списком. \*/* FractalGenerator mandelbrotFractal = new Mandelbrot();  
 myComboBox.addItem(mandelbrotFractal);  
 FractalGenerator tricornFractal = new Tricorn();  
 myComboBox.addItem(tricornFractal);  
 FractalGenerator burningShipFractal = new BurningShip();  
 myComboBox.addItem(burningShipFractal);  
  
 */\*\* Экземпляр ButtonHandler в поле со списком. \*/* ButtonHandler fractalChooser = new ButtonHandler();  
 myComboBox.addActionListener(fractalChooser);  
  
 */\*\*  
 \* Создает новый объект JPanel, JLabel, JComboBox, и добавляет  
 \* панель в кадр в положении NORTH.  
 \*/* JPanel myPanel = new JPanel();  
 JLabel myLabel = new JLabel("Fractal:");  
 myPanel.add(myLabel);  
 myPanel.add(myComboBox);  
 myFrame.add(myPanel, BorderLayout.*NORTH*);  
  
 */\*\*  
 \* Создет кнопку сохранения, добавляет кнопку в JPanel в положение  
 \* BorderLayout.SOUTH вместе с кнопкой reset.  
 \*/* saveButton = new JButton("Save");  
 JPanel myBottomPanel = new JPanel();  
 myBottomPanel.add(saveButton);  
 myBottomPanel.add(resetButton);  
 myFrame.add(myBottomPanel, BorderLayout.*SOUTH*);  
  
 */\*\* Экземпляр ButtonHandler для кнопки сохранения. \*/* ButtonHandler saveHandler = new ButtonHandler();  
 saveButton.addActionListener(saveHandler);  
  
  
 */\*\*  
 \* Показывает содержимое кадра, делает его видимым и запрещает  
 \* менять размер окна.  
 \*/* myFrame.pack();  
 myFrame.setVisible(true);  
 myFrame.setResizable(false);  
  
 }  
  
 */\*\*  
 \* Private метод для отображения фрактала. Этот метод смотрит  
 \* каждый пиксель на экране и вычисляет количество итераций  
 \* для соответствующих координат в области отображения фрактала.  
 \* Если количество итераций равно -1, устанавливается чёрный цвет пикселя.  
 \* Иначе выбирается значение, основанное на количесвте итераций.  
 \*/* private void drawFractal()  
 {  
 */\*\* Вызывает enableUI(false)? чтобы отключить все элементы управления UI во время рисования. \*\*/* enableUI(false);  
  
 */\*\* Устанавливает оставшиеся строки, на общее количество строк. \*\*/* rowsRemaining = displaySize;  
  
 */\*\* Цикл вызывает FractalWorker на каждой стороке дисплея и рисует фрактал \*/* for (int x=0; x<displaySize; x++){  
 FractalWorker drawRow = new FractalWorker(x);  
 drawRow.execute();  
 }  
  
 }  
 */\*\*  
 \* Включает или выключает кнопки интерфейса и список на основе указанного значения.  
 \* Обновляет включенное состояние кнопки "save", "reset" и списка  
 \*/* private void enableUI(boolean val) {  
 myComboBox.setEnabled(val);  
 resetButton.setEnabled(val);  
 saveButton.setEnabled(val);  
 }  
  
 */\*\* Внутренний класс для обработки событий ActionListener с помощью кнопки reset. \*/* private class ButtonHandler implements ActionListener  
 {  
 public void actionPerformed(ActionEvent e)  
 {  
 */\*\* Получает источник движения. \*/* String command = e.getActionCommand();  
  
 */\*\*  
 \* Если источником является поле со списком, то выводится,  
 \* выбранный пользователем фрактал.  
 \*/* if (e.getSource() instanceof JComboBox) {  
 JComboBox mySource = (JComboBox) e.getSource();  
 fractal = (FractalGenerator) mySource.getSelectedItem();  
 fractal.getInitialRange(range);  
 drawFractal();  
  
 }  
 */\*\*  
 \* Если источником является кнопка reset, то выводится  
 \* начальное положение фрактала.  
 \*/* else if (command.equals("Reset")) {  
 fractal.getInitialRange(range);  
 drawFractal();  
 }  
 */\*\*  
 \* Если источником является кнопка save, то сохраняется  
 \* текущий фрактал.  
 \*/* else if (command.equals("Save")) {  
  
 */\*\* Позволяет пользователю выбирать файл для сохранения. \*/* JFileChooser myFileChooser = new JFileChooser();  
  
 */\*\* Сохраняет только PNG изображения. \*/* FileFilter extensionFilter =  
 new FileNameExtensionFilter("PNG Images", "png");  
 myFileChooser.setFileFilter(extensionFilter);  
  
 */\*\* Проводник не позволяет выбирать не PNG файлы. \*/* myFileChooser.setAcceptAllFileFilterUsed(false);  
  
 */\*\*  
 \* Появляется окно "Сохранить файл", которое позволяет  
 \* пользователю выбрать каталог и файл для сохранения.  
 \*/* int userSelection = myFileChooser.showSaveDialog(display);  
  
 */\*\*  
 \* Если результатом операции выбора файла является  
 \* APPROVE\_OPTION, продолжает операцию сохранения файла.  
 \*/* if (userSelection == JFileChooser.*APPROVE\_OPTION*) {  
  
 */\*\* Получает файл и его имя. \*/* java.io.File file = myFileChooser.getSelectedFile();  
 String file\_name = file.toString();  
  
 */\*\* Пытается сохранить фрактал на диск. \*/* try {  
 BufferedImage displayImage = display.getImage();  
 javax.imageio.ImageIO.*write*(displayImage, "png", file);  
 }  
 */\*\*  
 \* Выдаеёт сообщение с ошибкой  
 \* если найдено исключение.  
 \* \*/* catch (Exception exception) {  
 JOptionPane.*showMessageDialog*(display,  
 exception.getMessage(), "Cannot Save Image",  
 JOptionPane.*ERROR\_MESSAGE*);  
 }  
 }  
 */\*\*  
 \* Если результатом операции выбора файла не является  
 \* APPROVE\_OPTION, возвращает значение.  
 \*/* else return;  
 }  
 }  
 }  
  
 */\*\* Внутренний класс для обработки событий MouseListener с экрана. \*/* private class MouseHandler extends MouseAdapter  
 {  
 */\*\*  
 \* Когда обработчик получает событие в виде нажатия клавиши мыши,  
 \* он показывает пиксельные координаты щелчка в области фрактала,  
 \* затем выводит и вызывает метод генератора recenterAndZoomRange()  
 \* с координатами, по которым щёлкнули, и масштабом 0,5.  
 \*/* @Override  
 public void mouseClicked(MouseEvent e)  
 {  
 */\*\* Возвращает, если оставшиеся строки не равны 0. \*\*/* if (rowsRemaining != 0) {  
 return;  
 }  
 */\*\* Получает x координаты области экрана при клике мыши. \*/* int x = e.getX();  
 double xCoord = fractal.*getCoord*(range.x,  
 range.x + range.width, displaySize, x);  
  
 */\*\* Получает y координаты области экрана при клике мыши. \*/* int y = e.getY();  
 double yCoord = fractal.*getCoord*(range.y,  
 range.y + range.height, displaySize, y);  
  
 */\*\*  
 \* Вызывает метод генератора recenterAndZoomRange()  
 \* с координатами, по которым щёлкнули, и масштабом 0,5.  
 \*/* fractal.recenterAndZoomRange(range, xCoord, yCoord, 0.5);  
  
 */\*\*  
 \* Перерисовывает фрактал после изменения  
 \* отображаемой области.  
 \*/* drawFractal();  
 }  
 }  
  
 */\*\* Вычисляет значение цвета для одной строки фрактала. \*/* private class FractalWorker extends SwingWorker<Object, Object>  
 {  
 */\*\* Поле для целочисленной y-координаты строки, что будет вычислена. \*/* int yCoordinate;  
  
 */\*\*  
 \* Целочисленный массив, хранящий, вычисленные значения RGB,  
 \* для каждого пикселя в строке.  
 \*/* int[] computedRGBValues;  
  
 */\*\* Конструктор берёт координату y, как аргумент и сохраняет координату. \*/* private FractalWorker(int row) {  
 yCoordinate = row;  
 }  
  
 */\*\*  
 \* Этот метод вызывается в фоновом потоке. Он вычисляет RGB значение  
 \* для всех пикселей в 1 строке и сохраняет их в соответствующий  
 \* элемент целочисленного массива. Возвращает null.  
 \*/* protected Object doInBackground() {  
  
 computedRGBValues = new int[displaySize];  
  
 */\*\* Берёт все пиксели в строке. \*/* for (int i = 0; i < computedRGBValues.length; i++) {  
  
 */\*\*  
 \* Находит соответствующие координаты xCoord И yCoord в области экрана фрактала.  
 \*/* double xCoord = fractal.*getCoord*(range.x,  
 range.x + range.width, displaySize, i);  
 double yCoord = fractal.*getCoord*(range.y,  
 range.y + range.height, displaySize, yCoordinate);  
  
 */\*\* Вычисляет номер итераций для координат в области экрана фрактала. \*/* int iteration = fractal.numIterations(xCoord, yCoord);  
  
 */\*\*  
 \* Если номер итерации = -1, устанавливает текущее значение  
 \* в вычисляемом массиве RGB в черный цвет.  
 \*/* if (iteration == -1){  
 computedRGBValues[i] = 0;  
 }  
  
 else {  
 */\*\* Иначе выбирает значение оттенка на основе числа отераций. \*/* float hue = 0.7f + (float) iteration / 200f;  
 int rgbColor = Color.*HSBtoRGB*(hue, 1f, 1f);  
  
 */\*\* Обновляет целочисленный массив с цветом для текущего пикселя. \*/* computedRGBValues[i] = rgbColor;  
 }  
 }  
 return null;  
  
 }  
 */\*\*  
 \* Вызывается, когда фоновая задача выполняется. Рисует пиксели для  
 \* текущей строки и обновляет дисплей для этой строки.  
 \*/* protected void done() {  
 */\*\*  
 \* Выполняет итерацию по массиву данных строк, рисует пиксели,  
 \* которые были вычислены в doInBackground90. Перерисовывает строку,  
 \* которая была изменена.  
 \*/* for (int i = 0; i < computedRGBValues.length; i++) {  
 display.drawPixel(i, yCoordinate, computedRGBValues[i]);  
 }  
 display.repaint(0, 0, yCoordinate, displaySize, 1);  
  
 */\*\* Уменьшает оставшиеся строки. Если 0, то вызывает enableUI(true). \*/* rowsRemaining--;  
 if (rowsRemaining == 0) {  
 enableUI(true);  
 }  
 }  
 }  
  
 */\*\*  
 \* Статический метод для запуска FractalExplorer. Запускает  
 \* новый экземпляр FractalExplorer с размером экрана 600,  
 \* вызывает createAndShowGUI() для обекта проводника и вызывает  
 \* drawFractal() для проводника, чтобы увидеть начальный вид.  
 \*/* public static void main(String[] args)  
 {  
 FractalExplorer displayExplorer = new FractalExplorer(600);  
 displayExplorer.createAndShowGUI();  
 displayExplorer.drawFractal();  
 }  
}

**4 Работа программы**

Время до появления фрактала на экране значительно уменьшилось, и теперь отрисовку фрактала можно наблюдать в процессе ее выполнения, чего нельзя сказать об однопоточной программе, где приходилось несколько секунд наблюдать черный экран.





