Московский Государственный Технический Университет

имени Н.Э. Баумана

Факультет ИУ «Информатика и системы управления»

Кафедра ИУ-3 «Информационные системы и телекоммуникации»

Отчет
к Практикуму №6

«Многозадачность в программе FractalExplorer»

по дисциплине «Технологии программирования»

Продолжительность работы: 2 ак. часа

Сдала Магомедов В.О.

Приняла Руденкова Ю.С.

Москва, 2020

Задание:

Я закончил работу над программой для изучения фракталов, добавив ей несколько новых свойств, включая возможность рисовать выбранный фрактал в нескольких фоновых потоках. Это дает два важных преимущества: первое, пользовательский интерфейс не зависает во время прорисовки нового фрактала, и второе, если компьютер имеет несколько процессоров, прорисовка будет выполняться гораздо быстрее. Многозадачные приложения могут быть довольно сложными, но Swing имеет встроенную поддержку управления фоновыми потоками.

До сих пор все вычисления фрактала производились в потоке диспетчере событий Swing. Этот поток обрабатывает все события Swing: нажатие кнопок, перерисовку, и так далее. Именно поэтому пользовательский интерфейс зависает, когда вычисляется фрактал; так как вычисления производятся в потоке диспетчере событий, события не могут обрабатываться до тех пор, пока вычисления не будут закончены.

Я изменил программу так, чтобы для вычисления фракталов использовался один или несколько фоновых потоков. Поток диспетчер событий не используется для вычисления фрактала. Если вычисление будет производиться в нескольких потоках, процедура вычисления разбивается на несколько независимых частей. Это очень просто сделать – можно дать каждому потоку для вычислений одну строчку фрактала.

Это в действительности весьма частая ситуация, возникающая при проектировании пользовательского интерфейса, пользовательский интерфейс должен инициировать длительную операцию, но операция может исполняться в фоновом потоке в то время как пользовательский интерфейс продолжает обслуживать запросы пользователя. Для того чтобы упростить этот способ взаимодействия с пользователем, Swing предлагает класс javax.swing.SwingWorker, который существенно упрощает управление исполнением фоновых потоков. SwingWorker это абстрактный класс; Я расширил его свойства, и добавил функции, необходимые для исполнения потока. Вот наиболее важные методы, которые потребовалось реализовать:

doInBackground() – этот метод исполняет фоновую операцию. Swing вызывает этот метод в фоновом потоке, а не в потоке диспетчере сообщений.

done() – этот метод вызывается, когда завершается исполнение фоновой операции. Он вызывается в контексте потока диспетчера событий, и следовательно из него доступны компоненты пользовательского интерфейса.

В документации класса SwingWorker есть одна неясность. В действительности это класс SwingWorker. Тип T это тип значения возвращаемого doInBackground(), когда исполнение задачи заканчивается. Тип V используется, если фоновая задача передает промежуточные результаты во время работы; эти промежуточные результаты выводятся методами publish() и process(). Использование одного или обоих этих типов не обязательно; если они

не используются, можно просто задать Object для этих типов.

Прорисовка в фоновом режиме

Надо было создать дочерний класс SwingWorker и дать ему имя FractalWorker. Этот класс является внутренним классом FractalExplorer. Это наиболее простой способ решить задачу, так как из методов класса понадобится доступ к некоторым внутренним членам класса FractalExplorer.

Так как SwingWorker использует технологию обобщенного программирования, надо было при объявлении класса указать его параметры - Object для обоих, потому что мне они не нужны. В конце концов, получится такое объявление класса: private class FractalWorker extends SwingWorker

Класс FractalWorker отвечает за вычисление цветов точек одной строки фрактала, и для этого ему создала 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 сохраняет ее.

Метод doInBackground() вызывается в фоновом потоке, и выполняет длительную во времени операцию. Взяла код для этого метода из старой функции “прорисовки фрактала”. Конечно, вместо того чтобы рисовать фрактал на экране, в цикле сохраняю каждое RGB значение в соответствующий элемент массива целых чисел. Я не могу изменять изображение на экране из этого потока, потому что это нарушает требования Swing!

Вместо этого, выделил память под массив целых чисел в начале этого метода, и затем записал в этот массив цвет каждой точки. Код, который я написал раньше очень легко привести к нужному виду – разница в том что теперь я вычисляю фрактал только для одной заданной строки и не изменяю изображение на экране.

Метод doInBackground() просто возвращает null! Метод done() вызывается когда фоновая задача завершена, и этот метод вызывается в контексте потока диспетчера событий Swing. Это означает, что я могу модифицировать компоненты Swing как угодно. Поэтому, здесь, я могу просто перебрать все точки строки, и прорисовать на экране их цвета вычисленные в doInBackground().

Как и раньше, после завершения прорисовки строки, необходимо попросить Swing перерисовать часть экрана, которая была изменена. Так как я изменил только одну строку, воспользуюсь вариантом JComponent.repaint() который позволяет задать границы области перерисовки.

Метод имеет небольшую странность – у него есть неиспользуемый параметр типа long в начале списка аргументов, просто подставил ему значение 0. В остальном, просто указываю строку, которую надо перерисовать – позицию (0, y) и размер (displaySize, 1).

Завершив разработку класса фоновой задачи, встроила его в процесс прорисовки фрактала. Я уже переместил часть своего кода из функции

"прорисовки фрактала" в класс SwingWorker, теперь функцию "прорисовки фрактала" надо изменить:

Для каждой строчки фрактала, создал отдельный объект SwingWorker и затем вызвала его метод execute(). Этот метод создаст фоновый поток и запустит его на исполнение!

После того как я написал и отладил этот функционал, интерфейс стал прорисовываться и откликаться на команды гораздо быстрее. Однако, если я кликну по экрану или по кнопке во время прорисовки изображения, программа начнет обрабатывать событие, хотя логично было бы игнорировать его до тех пор, пока не будет завершена текущая операция. К счастью это довольно легко исправить.

Блокировка событий во время перерисовки

Легче всего решить эту проблему обработки событий во время перерисовки изображения можно следя за количеством строк, которые осталось прорисовать, и игнорируя события от пользовательского интерфейса пока все они не будут перерисованы. Однако делать это следует осторожно, избегая возможных ошибок.

Сделал так: добавила поле "количество оставшихся строк" к классу Fractal Explorer, и использую его для определения конца вычислений. Буду считывать и изменять значение этого поля только в потоке диспетчере событий, чтобы избежать параллельного доступа к полю из нескольких потоков. Если я буду работать с полем только из одного потока, то смогу избежать различных ошибок, связанных с параллельным доступом. Вот что я сделал:

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

Функция "прорисовки фрактала" стала делать еще две дополнительные вещи. Во-первых, метод enableUI(false) вызвается в самом ее начале, для того чтобы заблокировать все элементы пользовательского интерфейса перед прорисовкой. Во-вторых, установила значение "количества оставшихся строк" равным общему числу строк изображения. Сделал это перед тем как запустить фоновые потоки прорисовки, иначе все это будет работать неправильно!

В методе done() фонового потока, уменьшил "количество отавшийся строк" на единицу как завершающий шаг операции. Затем, если количество оставшихся строк становится равным 0, вызываю enableUI(true). Иначе говоря, я буду отвечать на события от мыши только после того как все строки будут прорисованы.

Код:

Изменим файл FractalExplorer:

```
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.io.FileNotFoundException;
import java.io.IOException;

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;

/**
 * Этот класс предоставляет графический интерфейс для отображения фракталов.
 */
public class FractalExplorer {
    /** Длина стороны квадратной области дисплея.**/
    private int dispSize;
    /** Область изображения для  фрактала. **/
    private JImageDisplay img;
    /** Используется для генерации фракталов определенного вида. **/
    private FractalGenerator fGen;
    /** Задает отображаемый диапазон в комплексной области.**/
    private Rectangle2D.Double range;
    /** Различные компоненты графического интерфейса. **/
    JFrame frame;
    JButton resetButton;
    JButton saveButton;
    JLabel label;
    JComboBox<FractalGenerator> fractalCBox;
    JPanel cbPanel;
    JPanel buttonPanel;
    /** Количество строк, которые все еще рисуются. **/
    int rowsRemaining;

    /** Базовый конструктор. Инициализирует отображение изображения, генератор
фракталов,и начальная зона просмотра.
     */
    public FractalExplorer(int dispSize) {
        this.dispSize = dispSize;
        this.fGen = new Mandelbrot();
        this.range = new Rectangle2D.Double(0, 0, 0, 0);
        fGen.getInitialRange(this.range);

    }

    /**
     * Настройка и отображение графического интерфейса пользователя.
     */
    public void createAndShowGUI() {
// Создание компонентов графического интерфейса.
        frame = new JFrame("Fractal Explorer");
        img = new JImageDisplay(dispSize, dispSize);
        resetButton = new JButton("Reset Display");
```

```
        resetButton.setActionCommand("reset");
        saveButton = new JButton("Save Image");
        saveButton.setActionCommand("save");
        label = new JLabel("Fractal: ");
        fractalCBox    = new JComboBox<FractalGenerator>();
        cbPanel = new JPanel();
        cbPanel.add(label);
        cbPanel.add(fractalCBox);
        buttonPanel = new JPanel();
        buttonPanel.add(saveButton);
        buttonPanel.add(resetButton);
        fractalCBox.addItem(new Mandelbrot());
        fractalCBox.addItem(new BurningShip());
        fractalCBox.addItem(new Tricorn());

        //Реакции
        ActionHandler aHandler = new ActionHandler();
        MouseHandler mHandler = new MouseHandler();
        resetButton.addActionListener(aHandler);
        saveButton.addActionListener(aHandler);
        img.addMouseListener(mHandler);
        fractalCBox.addActionListener(aHandler);

        // Рамка
        frame.setLayout(new java.awt.BorderLayout());
        frame.add(img, java.awt.BorderLayout.CENTER);
        frame.add(buttonPanel, java.awt.BorderLayout.SOUTH);
        frame.add(cbPanel, java.awt.BorderLayout.NORTH);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Показываем изображение
        frame.pack();
        frame.setVisible(true);
        frame.setResizable(false);
    }

    /** Включение или отключение интерактивности пользовательского интерфейса. **/
    public void enableUI(boolean val) {
        saveButton.setEnabled(val);
        resetButton.setEnabled(val);
        fractalCBox.setEnabled(val);
    }

    /** Используем генератор фракталов, чтобы нарисовать фрактал ряд за рядом. **/
    private void drawFractal() {
        /** Отключаем интерактивность пользовательского интерфейса во время рисования.
**/
        enableUI(false);
        // Рисуем фрактал линия за линией.
        rowsRemaining = dispSize;
        for (int i = 0; i < dispSize; i++) {
            FractalWorker rowDrawer = new FractalWorker(i);
            rowDrawer.execute();
        }
    }

    /** Запуск приложения. **/
    public static void main(String[] args) {
        FractalExplorer fracExp = new FractalExplorer(800);
        fracExp.createAndShowGUI();
        fracExp.drawFractal();
    }

    /**
     * Сброс масштаба, изменения фрактального типа, сохранение изображений
     **/
```

```
public class ActionHandler implements ActionListener {
    /** Действия.**/
    public void actionPerformed(ActionEvent e) {
        /**  Если нажата кнопка сброса, сбрасываем масштаб. **/
        if (e.getActionCommand() == "reset") {
            fGen.getInitialRange(range);
            drawFractal();
        }
        /**  Если нажимаем кнопку Сохранить, сохраняем изображение. **/
        else if (e.getActionCommand() == "save") {
            JFileChooser fileChooser = new JFileChooser();
            FileFilter filter
                    = new FileNameExtensionFilter("PNG Images", "png");
            fileChooser.setFileFilter(filter);
            fileChooser.setAcceptAllFileFilterUsed(false);
            int res = fileChooser.showSaveDialog(img);

            if (res == JFileChooser.APPROVE_OPTION) {
                try {
                    javax.imageio.ImageIO.write(img.getBufferedImage(),
                            "png", fileChooser.getSelectedFile());
                } catch (NullPointerException | IOException e1) {
                    javax.swing.JOptionPane.showMessageDialog(img,
                            e1.getMessage(), "Cannot Save Image",
                            JOptionPane.ERROR_MESSAGE);
                }
            }
            else {
                return;
            }
        }
        /** Если нажимаем кнопку combobox, изменяем фрактальные типы.  **/
        else if (e.getSource() == (Object) fractalCBox) {
            fGen = (FractalGenerator) fractalCBox.getSelectedItem();
            fGen.getInitialRange(range);
            drawFractal();
        }
    }
}


/** Увеличиваем масштаб **/
class MouseHandler extends MouseAdapter {
    @Override
    public void mouseClicked(MouseEvent e) {
        // Если строки рисуются, игнорируем ввод с помощью мыши.
        if (rowsRemaining != 0) {
            return;
        }
        double xCoord = FractalGenerator.getCoord(range.x,
                range.x + range.width, dispSize, e.getX());
        double yCoord = FractalGenerator.getCoord(range.y,
                range.y + range.width, dispSize, e.getY());
        fGen.recenterAndZoomRange(range, xCoord, yCoord, 0.5);
        drawFractal();
    }
}

/** Этот класс рисует одну строку фрактала. **/
private class FractalWorker extends SwingWorker<Object, Object> {
    /** Y-координата для строки **/
    int rowY;
    /** Массив значений rgb для пикселей в этой строке. **/
    int[] rgbVals;

    /** Конструктор **/
```

```
        public FractalWorker(int yCoord) {
            rowY = yCoord;
        }

        /**
         * Вычисляет значения rgb для строки и сохраняет их в памяти массива.
         **/
        public Object doInBackground() {
            rgbVals = new int[dispSize];
            double yCoord = FractalGenerator.getCoord(range.y,
                    range.y + range.width, dispSize, rowY);

            for (int i = 0; i < dispSize; i++) {
                double xCoord = FractalGenerator.getCoord(range.x,
                        range.x + range.width, dispSize, i);
                double numIters = fGen.numIterations(xCoord, yCoord);

                if (numIters == -1) {
                    /** Пикселя в наборе нет. Покрасим его в черный цвет. **/
                    rgbVals[i] = 0;
                }
                else {
                    /**Пиксель находится во фрактальном множестве.
                     * Раскрасим пиксель в зависимости от количества итераций которые
потребовались
                     */
                    float hue = 0.7f + (float) numIters / 200f;
                    int rgbColor = Color.HSBtoRGB(hue, 1f, 1f);
                    rgbVals[i] = rgbColor;
                }
            }
            return null;
        }

        /**
         * Выводит значения rgb на экран.
         */
        public void done() {
            for (int i = 0; i < dispSize; i++) {
                img.drawPixel(i, rowY, rgbVals[i]);
            }
            img.repaint(0, 0, rowY, dispSize, 1);
            // Уменьшаем количество строк, которые все еще рисуются.
            rowsRemaining -= 1;
            // // Если все строки сделаны, повторно включим пользовательский интерфейс.
            if (rowsRemaining == 0) {
                enableUI(true);
            }
        }
    }
}
```

Скриншоты:

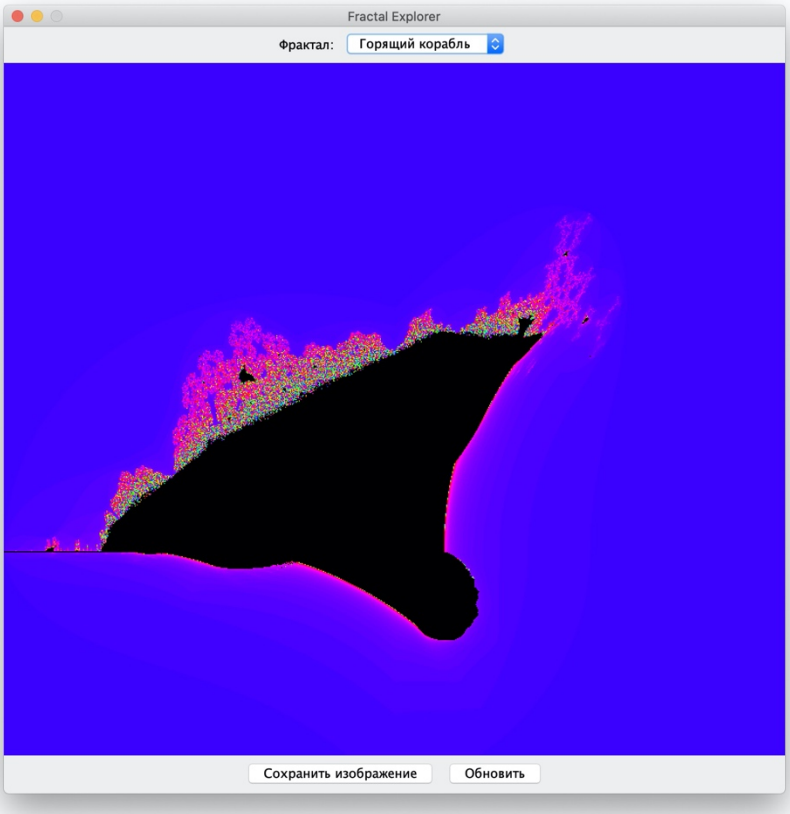

Рис.1 Увеличенное изображение фрактала «горящий корабль»

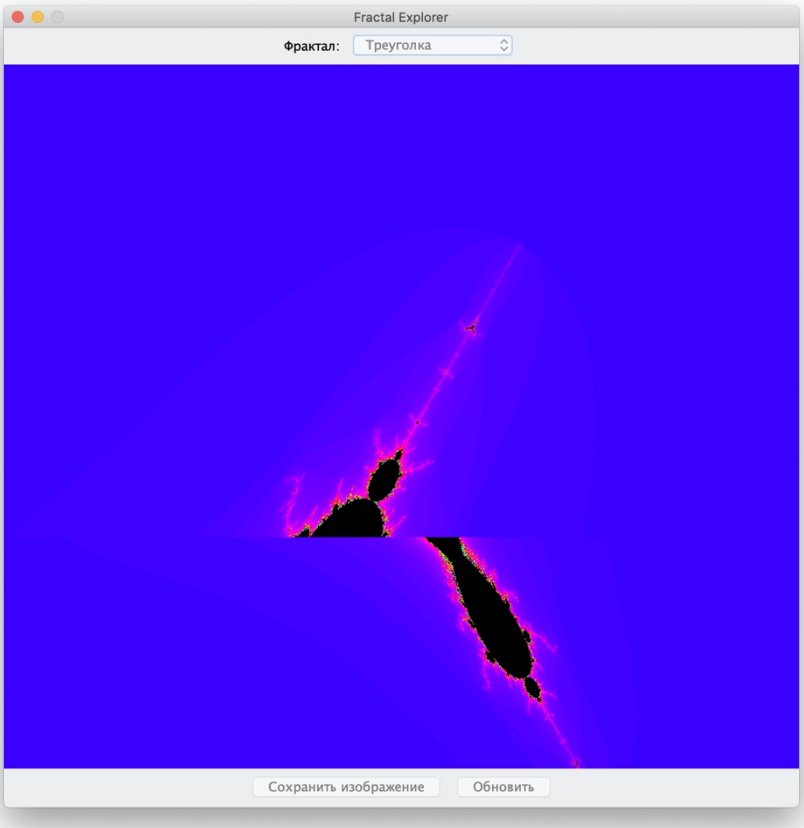

Рис.2 Загрузка фрактала

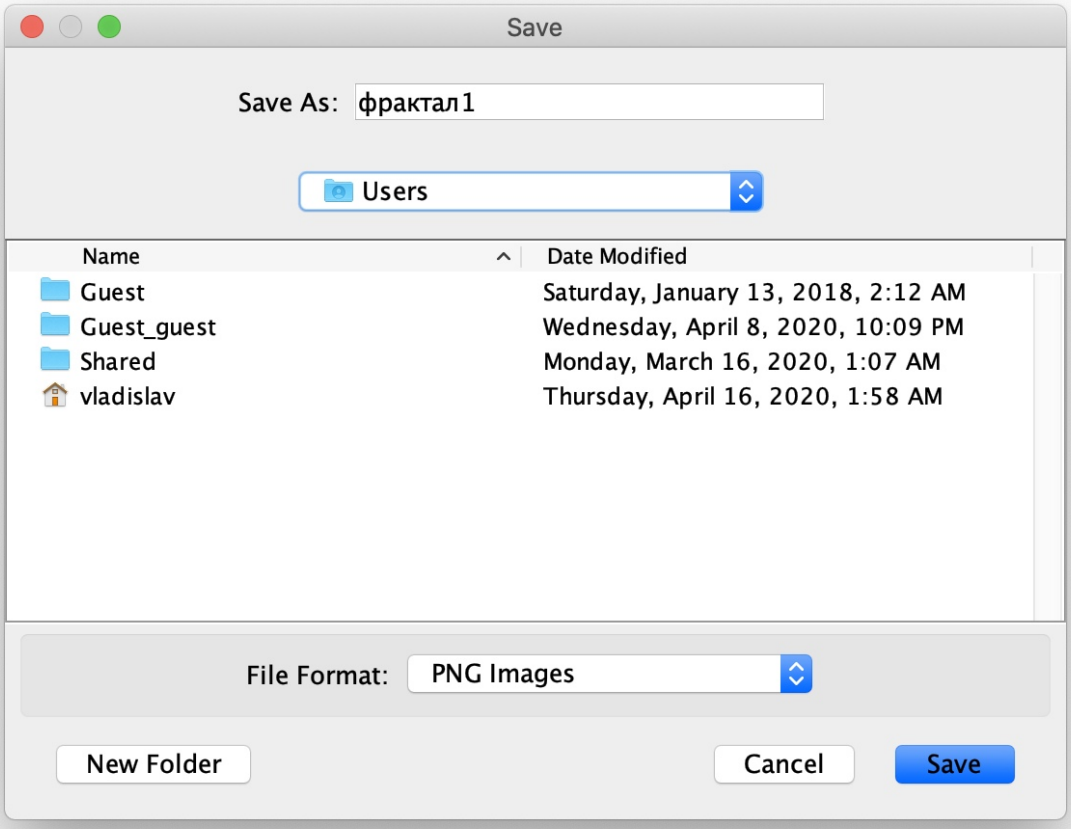

Рис.3 Сохранение изображения фрактала

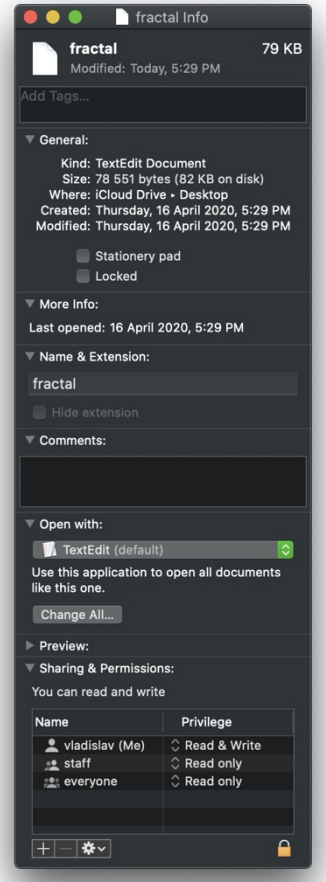

Рис.4 Свойства изображения фрактала

Вывод:

В этой лабораторной работе был изменен файл FractalExplorer. Мы добавили несколько новых свойств, включая возможность рисовать выбранный фрактал в нескольких фоновых потоках. Добавление новых свойств дало два важных преимущества: пользовательский интерфейс не зависает во время прорисовки нового фрактала, и, если компьютер имеет несколько процессоров, прорисовка будет выполняться гораздо быстрее.

Завершив работу, я получил красивую и надежную программу для исследования, которая может прорисовывать изображение в нескольких потоках, и не позволяет пользователю вмешаться, пока процесс вычисления фрактала в фоновом режиме не закончится.