Федеральное агентство связи

Ордена трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский технический университет связи и информатики»

Кафедра Математической кибернетики и информационных технологий

Отчет по лабораторной работе № 6 «Многопоточный генератор фракталов»

по дисциплине «Объектно-ориентированное программирование»

Выполнил: студент группы БФИ1901

Кириллов Роман Сергеевич

Проверил: Мосева М.С

Москва, 2020

Задача:

Необходимо будет работать в классе FractalExplorer. Часть кода будет новой, но некоторые части будут представлять из себя модифицированный код, который вы уже написали.

1) Создайте подкласс SwingWorker с именем FractalWorker, который будет внутренним классом FractalExplorer. Это необходимо для того, чтобы у него был доступ к нескольким внутренним членам FractalExplorer. Помните, что класс SwingWorker является универсальным, поэтому нужно указать параметры - можно просто указать Object для двух параметров, потому что в данной реализации эти параметры не будут использоваться. В результате у вас должна получиться следующая строчка кода: private class FractalWorker extends SwingWorker

2) Класс FractalWorker будет отвечать за вычисление значений цвета для одной строки фрактала, поэтому ему потребуются два поля: целочисленная yкоордината вычисляемой строки, и массив чисел типа int для хранения вычисленных значений RGB для каждого пикселя в этой строке. Конструктор должен будет получать y-координату в качестве параметра и сохранять это. (На данном этапе не надо выделять память под целочисленный массив, так как он не потребуется, пока строка не будет вычислена.)

3) Метод doInBackground() вызывается в фоновом потоке и отвечает за выполнение длительной задачи. Поэтому в вашей реализации вам нужно будет взять часть кода из вашей предыдущей функции «draw fractal» и поместить ее в этот метод. Вместо того, чтобы рисовать изображение в окне, цикл должен будет сохранить каждое значение RGB в соответствующем элементе целочисленного массива. Вы не сможете изменять отображение из этого потока, потому что вы нарушите ограничения ограничения потоков Swing.

4) Вместо этого выделите память для массив целых чисел в начале реализации этого метода (массив должен быть достаточно большим для хранения целой строки значений цвета), а затем сохраните цвет каждого пикселя в этом массиве. Единственные различия между настоящим и предыдущим кодом в том, что вам нужно будет вычислить фрактал для указанной строки, и что вы на данном этапе не обновляете отображение. Метод doInBackground() должен возвращать объект типа Object, так как это указано в объявлении SwingWorker . Просто верните null.

5) Метод done() вызывается, когда фоновая задача завершена, и этот метод вызывается из потока обработки событий Swing. Это означает, что вы можете модифицировать компоненты Swing на ваш вкус. Поэтому в этом методе вы можете перебирать массив строк данных, рисуя пиксели, которые были вычислены в doInBackground (). После того, как строка будет вычислена, вам нужно будет сообщить Swing, перерисовать часть изображения, которая была изменена. Поскольку вы изменили только одну строку, перерисовывать изображение целиком будет затратно, поэтому вы можете использовать метод JComponent.repaint(), который позволит вам указать область для перерисовки. У данного метода есть неиспользуемый параметр типа long, вы можете просто указать 0 для этого аргумента. В качестве остальных параметров укажите вычисленную строку, значения начала фрагмента для перерисовки (0, y) и конечные значения фрагмента (displaySize, 1). После того, как вы завершили класс для фоновой задачи, следующим шагом нужно будет привязать его к процессу рисования фракталов. Так как часть кода из функции «draw fractal» уже задействована в разрабатываемом классе, на данном этапе можно изменить функцию «draw fractal», а именно, для каждой строки в отображении создать отдельный рабочий объект, а затем вызвать для него метод execute (). Это действие запустит фоновый поток и запустит задачу в фоновом режиме. Помните, что класс FractalWorker отвечает за генерацию данных строки и за рисование этой строки, поэтому функция «draw fractal» должна быть простой.

Создайте функцию void enableUI(boolean val), которая будет включать или отключать кнопки с выпадающим списком в пользовательском интерфейсе на основе указанного параметра. Для включения или отключения этих компонентов можно использовать метод Swing setEnabled(boolean). Убедитесь, что ваш метод обновляет состояние кнопки сохранения, кнопки сброса и выпадающего списка.

• Функция «draw fractal» должна сделать еще две вещи. Первая - она должна вызвать метод enableUI (false), чтобы отключить все элементы пользовательского интерфейса во время рисования. Вторая - она должна установить значение «rows remaining» равным общему количеству строк, которые нужно нарисовать. Эти действия должны быть сделаны перед выполнением каких-либо рабочих задач, иначе это может привести к некорректной работе алгоритма.

• В методе done(), уменьшите значение «rows remaining» на 1, как последний шаг данной операции. Затем, если после уменьшения значение «rows remaining» равно 0, вызовите метод enableUI (true).

• Наконец, измените реализацию mouse-listener для того, чтобы она сразу возвращалась в предыдущее состояние, если значение «rows remaining» не равно нулю. Другими словами, приложение будет реагировать на щелчки мышью, только в том случае, если больше нет строк, которые должны быть нарисованы. (Обратите внимание, что также не нужно вносить аналогичные изменения в обработчике событий, потому что все эти компоненты будут отключены с помощью метода enableUI ().)

Ход работы:

Класс Mandelbrot

import java.awt.geom.Rectangle2D;  
public class Mandelbrot extends FractalGenerator {  
 public static final int *MAX\_ITERATIONS* = 2000;  
 public void getInitialRange(Rectangle2D.Double range) {  
 range.x = -2;  
 range.y = -1.5;  
 range.width = 3;  
 range.height = 3;  
 }  
 public int numIterations(double x, double y) {  
 double r = 0, i = 0, r1, i1;  
 int k = 0;  
 while (k < *MAX\_ITERATIONS* && (r \* r + i \* i) < 4) {  
 r1 = r \* r - i \* i + x;  
 i1 = 2 \* r \* i + y;  
 r = r1;  
 i = i1;  
 k++;  
 }  
 if (k == *MAX\_ITERATIONS*)  
 return -1;  
 return k;  
 }  
 @Override  
 public String toString() {  
 return "Mandelbrot";  
 }  
}

Класс JImageDisplay

import javax.swing.JComponent;  
import java.awt.Color;  
import java.awt.Dimension;  
import java.awt.Graphics;  
import java.awt.Graphics2D;  
import java.awt.image.BufferedImage;  
public class JImageDisplay extends JComponent {  
 private BufferedImage m;  
 public BufferedImage im() {  
 return m;  
 }  
 public JImageDisplay(int w, int h){  
 m = new BufferedImage(w, h, BufferedImage.*TYPE\_INT\_RGB*);  
 Dimension ty = new Dimension(w, h);  
 super.setPreferredSize(ty);  
 }  
 @Override  
 protected void paintComponent(Graphics g) {  
 super.paintComponent(g);  
 g.drawImage(m,0,0,m.getWidth(), m.getHeight(), null);  
 }  
 public void clearImage() {  
 Graphics2D itr = m.createGraphics();  
 itr.setColor(Color.*BLACK*);  
 itr.fillRect(0, 0, m.getWidth(), m.getHeight());  
 }  
 public void drawPixel (int x, int y, int rgbColor){  
 m.setRGB(x, y, rgbColor);  
 }  
}

Класс FractalGenerator

import java.awt.geom.Rectangle2D;  
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);  
}

FractalExplorer

import javax.imageio.ImageIO;  
import javax.swing.\*;  
import javax.swing.filechooser.FileNameExtensionFilter;  
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.BufferedImage;  
import java.io.File;  
import javax.swing.JFileChooser;  
  
public class FractalExplorer {  
 private int si;  
 private JImageDisplay JDisplay;  
 private FractalGenerator gen;  
 private Rectangle2D.Double d;  
  
 public FractalExplorer(int size) {  
 si = size;  
 gen = new Mandelbrot();  
 d = new Rectangle2D.Double();  
 gen.getInitialRange(d);  
 JDisplay = new JImageDisplay(si, si);  
 }  
  
 JComboBox combobox = new JComboBox();  
 public void createAndShowGUI() {  
 JDisplay.setLayout(new BorderLayout());  
 JFrame frame = new JFrame("Fractal Explorer");  
  
 JButton reset = new JButton("Reset Display");  
  
 reset rez = new reset();  
 reset.addActionListener((ActionListener) rez);  
  
 JPanel south = new JPanel();  
 FractalGenerator mand = new Mandelbrot();  
 FractalGenerator tric = new Tricorn();  
 FractalGenerator burn = new BurningShip();  
 JPanel north = new JPanel();  
  
  
 combobox.addItem(mand);  
 combobox.addItem(burn);  
 combobox.addItem(tric);  
  
 choose fractals= new choose();  
 combobox.addActionListener(fractals);  
  
 JLabel label = new JLabel("Fractal:");  
 north.add(label);  
 north.add(combobox);  
  
 JButton save = new JButton("Save");  
 save save1 = new save();  
 save.addActionListener(save1);  
 south.add(save);  
 south.add(reset);  
 kliker klik = new kliker();  
 JDisplay.addMouseListener(klik);  
  
 frame.add(north, BorderLayout.*NORTH*);  
 frame.add(south, BorderLayout.*SOUTH*);  
 frame.add(JDisplay, BorderLayout.*CENTER*);  
 frame.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
  
 frame.pack();  
 frame.setVisible(true);  
 frame.setResizable(false);  
 }  
  
  
 private void enableUI(boolean val){ /////////  
 combobox.setEnabled(val);  
 }  
 int si1;  
 private void drawFractal() {  
 enableUI (false);  
 si1 = si;  
 for (int y = 0; y < si; y++) {  
 FractalWorker st = new FractalWorker(y);  
 st.execute();  
 }  
 JDisplay.repaint();  
 }  
  
 private class FractalWorker extends SwingWorker<Object, Object>{  
 int y;  
 int [] rgb;  
  
 public FractalWorker(int y) {  
 this.y = y;  
 }  
  
 @Override  
 protected Object doInBackground() throws Exception {  
 double x1, y1;  
 rgb = new int [si];  
 for (int x = 0; x < si; x++) {  
 x1 = FractalGenerator.*getCoord*(d.x, d.x + d.width, si, x);  
 y1 = FractalGenerator.*getCoord*(d.y, d.y + d.width, si, y);  
 int k = gen.numIterations(x1, y1);  
 if (k == -1) {  
 rgb[x] = 0;  
 } else {  
 float hue = 0.7f + (float) k / 200f;  
 int rgbColor = Color.*HSBtoRGB*(hue, 1f, 1f);  
 rgb[x] = rgbColor;  
 }  
 }  
 return null;  
 }  
 @Override  
 protected void done() {  
 for(int x = 0; x<si; x++)  
 JDisplay.drawPixel(x, y, rgb[x]);  
 si1--;  
 JDisplay.repaint(0, 0,y,si,1);   
 if(si1 == 0){  
 enableUI (true);}  
 }  
 }  
  
 private class save implements ActionListener {  
 public void actionPerformed(ActionEvent e) {  
 if (e.getActionCommand().equals("Save")) {  
 JFileChooser chooser = new JFileChooser();  
 FileNameExtensionFilter filter = new FileNameExtensionFilter("PNG Images", "png");  
 chooser.setFileFilter(filter);  
 chooser.setAcceptAllFileFilterUsed(false);  
 if (chooser.showSaveDialog(JDisplay) == JFileChooser.*APPROVE\_OPTION*) {  
 java.io.File file = chooser.getSelectedFile();  
 String image = file.getPath();  
 file = new File(image + ".png");  
 try {  
 BufferedImage im = JDisplay.im();  
 ImageIO.*write*(im, "png", file);  
 } catch (Exception exception) {  
 JOptionPane.*showMessageDialog*(JDisplay, exception.getMessage(), "ERROR", JOptionPane.*ERROR\_MESSAGE*);  
 }  
 }  
 }  
 }  
 }  
  
 private class choose implements ActionListener {  
 public void actionPerformed(ActionEvent e) {  
 JComboBox comboBox = (JComboBox) e.getSource();  
 gen = (FractalGenerator) comboBox.getSelectedItem();  
 gen.getInitialRange(d);  
 drawFractal();  
 }  
 }  
  
 private class  
  
 reset implements ActionListener  
 {  
 public void actionPerformed(ActionEvent e)  
 {  
 gen.getInitialRange(d);  
 drawFractal();  
 }  
 }  
  
 private class kliker extends MouseAdapter  
 {  
 @Override  
 public void mouseClicked(MouseEvent e) {  
 if(si1==0) {  
 super.mouseClicked(e);  
 int x = e.getX();  
 int y = e.getY();  
 double x1, y1;  
 x1 = FractalGenerator.*getCoord*(d.x, d.x + d.width, si, x);  
 y1 = FractalGenerator.*getCoord*(d.y, d.y + d.height, si, y);  
 gen.recenterAndZoomRange(d, x1, y1, 0.5);  
 drawFractal();  
 }  
 }  
 }  
  
 public static void main(String[] args)  
 {  
 FractalExplorer displayExplorer = new FractalExplorer(700);  
 displayExplorer.createAndShowGUI();  
 displayExplorer.drawFractal();  
 }  
}

Класс BurningShip

import java.awt.geom.Rectangle2D;  
public class BurningShip extends FractalGenerator{  
 public static final int *MAX\_ITERATIONS* = 2000;  
 public void getInitialRange(Rectangle2D.Double range) {  
 range.x = -2;  
 range.y = -2.5;  
 range.width = 4;  
 range.height = 4;  
 }  
 public int numIterations(double x, double y) {  
 double r = 0, i = 0, r1, i1;  
 int k = 0;  
 while (k < *MAX\_ITERATIONS* && (r \* r + i \* i) < 4) {  
 r1 = (r \* r) - (i \* i) + x;  
 i1 = 2 \* Math.*abs*(r\*i) + y;  
 r = r1;  
 i = i1;  
 k++;  
 }  
 if (k == *MAX\_ITERATIONS*)  
 return -1;  
 return k;  
 }  
 @Override  
 public String toString() {  
 return "Burning Ship";  
 }  
}

Класс Tricorn

import java.awt.geom.Rectangle2D;  
public class Tricorn extends FractalGenerator{  
 public static final int *MAX\_ITERATIONS* = 2000;  
 public void getInitialRange(Rectangle2D.Double range) {  
 range.x = -2;  
 range.y = -2;  
 range.width = 4;  
 range.height = 4;  
 }  
 public int numIterations(double x, double y) {  
 double r = 0, i = 0, r1, i1;  
 int k = 0;  
 while (k < *MAX\_ITERATIONS* && (r \* r + i \* i) < 4) {  
 r1 = r \* r - i \* i + x;  
 i1 = - 2 \* r \* i + y;  
 r = r1;  
 i = i1;  
 k++;  
 }  
 if (k == *MAX\_ITERATIONS*)  
 return -1;  
 return k;  
 }  
 @Override  
 public String toString() {  
 return "Tricorn";  
 }  
}

Результат работы:







