МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ

КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ

Ордена Трудового Знамени федеральное государственное

Бюджетное образовательное учреждение высшего образования

МОСКОВСКИЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ СВЯЗИ И

ИНФОРМАТИКИ

Кафедра математической кибернетики и информационных технологий

Лабораторная работа № 6 по теме:

«Многопоточный генератор фракталов»

Выполнил: студент группы БИБ1902 \_\_\_\_\_\_\_Арустамян А.Б.

Проверил: \_\_\_\_\_\_\_\_Херсонский А. В.

Москва 2021

**Цель работы:**

В данной лабораторной работе необходимо будет реализовать возможность рисования фрактала с несколькими фоновыми потоками.

**Код программы:**

**FractalExplorer.java**

import java.awt.\*;  
import javax.swing.\*;  
import java.awt.geom.Rectangle2D;  
import java.awt.event.\*;  
import javax.swing.filechooser.\*;  
import java.awt.image.\*;  
  
public class FractalExplorer  
{  
 private JButton saveButton;  
 private JButton resetButton;  
 private JComboBox myComboBox;  
  
 private int rowsRemaining;  
  
 private int displaySize;  
  
 private JImageDisplay display;  
  
 private FractalGenerator fractal;  
  
 private Rectangle2D.Double range;  
  
 public FractalExplorer(int size) {  
  
 displaySize = size;  
  
 fractal = new Mandelbrot();  
 range = new Rectangle2D.Double();  
 fractal.getInitialRange(range);  
 display = new JImageDisplay(displaySize, displaySize);  
  
 }  
  
 public void createAndShowGUI()  
 {  
 display.setLayout(new BorderLayout());  
 JFrame myFrame = new JFrame("Fractal Explorer");  
  
 myFrame.add(display, BorderLayout.*CENTER*);  
  
 resetButton = new JButton("Reset");  
  
 ButtonHandler resetHandler = new ButtonHandler();  
 resetButton.addActionListener(resetHandler);  
  
 MouseHandler click = new MouseHandler();  
 display.addMouseListener(click);  
  
 myFrame.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
  
 myComboBox = new JComboBox();  
  
 FractalGenerator mandelbrotFractal = new Mandelbrot();  
 myComboBox.addItem(mandelbrotFractal);  
 FractalGenerator tricornFractal = new Tricorn();  
 myComboBox.addItem(tricornFractal);  
 FractalGenerator burningShipFractal = new BurningShip();  
 myComboBox.addItem(burningShipFractal);  
  
 ButtonHandler fractalChooser = new ButtonHandler();  
 myComboBox.addActionListener(fractalChooser);  
  
 JPanel myPanel = new JPanel();  
 JLabel myLabel = new JLabel("Fractal:");  
 myPanel.add(myLabel);  
 myPanel.add(myComboBox);  
 myFrame.add(myPanel, BorderLayout.*NORTH*);  
  
 saveButton = new JButton("Save");  
 JPanel myBottomPanel = new JPanel();  
 myBottomPanel.add(saveButton);  
 myBottomPanel.add(resetButton);  
 myFrame.add(myBottomPanel, BorderLayout.*SOUTH*);  
  
 ButtonHandler saveHandler = new ButtonHandler();  
 saveButton.addActionListener(saveHandler);  
  
 myFrame.pack();  
 myFrame.setVisible(true);  
 myFrame.setResizable(false);  
  
 }  
  
 private void drawFractal()  
 {  
 enableUI(false);  
  
 rowsRemaining = displaySize;  
  
 for (int x=0; x<displaySize; x++){  
 FractalWorker drawRow = new FractalWorker(x);  
 drawRow.execute();  
 }  
  
 }  
  
 private void enableUI(boolean val) {  
 myComboBox.setEnabled(val);  
 resetButton.setEnabled(val);  
 saveButton.setEnabled(val);  
 }  
  
 private class ButtonHandler implements ActionListener  
 {  
 public void actionPerformed(ActionEvent e)  
 {  
 String command = e.getActionCommand();  
  
 if (e.getSource() instanceof JComboBox) {  
 JComboBox mySource = (JComboBox) e.getSource();  
 fractal = (FractalGenerator) mySource.getSelectedItem();  
 fractal.getInitialRange(range);  
 drawFractal();  
  
 }  
 else if (command.equals("Reset")) {  
 fractal.getInitialRange(range);  
 drawFractal();  
 }  
 else if (command.equals("Save")) {  
  
 JFileChooser myFileChooser = new JFileChooser();  
  
 FileFilter extensionFilter =  
 new FileNameExtensionFilter("PNG Images", "png");  
 myFileChooser.setFileFilter(extensionFilter);  
  
 myFileChooser.setAcceptAllFileFilterUsed(false);  
  
 int userSelection = myFileChooser.showSaveDialog(display);  
  
 if (userSelection == JFileChooser.*APPROVE\_OPTION*) {  
  
 java.io.File file = myFileChooser.getSelectedFile();  
 String file\_name = file.toString();  
  
 try {  
 BufferedImage displayImage = display.getImage();  
 javax.imageio.ImageIO.*write*(displayImage, "png", file);  
 }  
 catch (Exception exception) {  
 JOptionPane.*showMessageDialog*(display,  
 exception.getMessage(), "Cannot Save Image",  
 JOptionPane.*ERROR\_MESSAGE*);  
 }  
 }  
  
 else return;  
 }  
 }  
 }  
  
 private class MouseHandler extends MouseAdapter  
 {  
 public void mouseClicked(MouseEvent e)  
 {  
 if (rowsRemaining != 0) {  
 return;  
 }  
 int x = e.getX();  
 double xCoord = fractal.getCoord(range.x,  
 range.x + range.width, displaySize, x);  
  
 int y = e.getY();  
 double yCoord = fractal.getCoord(range.y,  
 range.y + range.height, displaySize, y);  
  
 fractal.recenterAndZoomRange(range, xCoord, yCoord, 0.5);  
 drawFractal();  
 }  
 }  
  
 private class FractalWorker extends SwingWorker<Object, Object>  
 {  
  
 int yCoordinate;  
  
 int[] computedRGBValues;  
  
 private FractalWorker(int row) {  
 yCoordinate = row;  
 }  
  
 // метод, выполняющий фоновые операции  
 protected Object doInBackground() {  
  
 computedRGBValues = new int[displaySize];  
  
 for (int i = 0; i < computedRGBValues.length; i++) {  
  
 double xCoord = fractal.getCoord(range.x,  
 range.x + range.width, displaySize, i);  
 double yCoord = fractal.getCoord(range.y,  
 range.y + range.height, displaySize, yCoordinate);  
  
 int iteration = fractal.numIterations(xCoord, yCoord);  
  
 if (iteration == -1){  
 computedRGBValues[i] = 0;  
 }  
  
 else {  
 float hue = 0.7f + (float) iteration / 200f;  
 int rgbColor = Color.*HSBtoRGB*(hue, 1f, 1f);  
  
 computedRGBValues[i] = rgbColor;  
 }  
 }  
 return null;  
  
 }  
 // вызывается при завершении фоновой задачи  
 protected void done() {  
  
 for (int i = 0; i < computedRGBValues.length; i++) {  
 display.drawPixel(i, yCoordinate, computedRGBValues[i]);  
 }  
  
 display.repaint(0, 0, yCoordinate, displaySize, 1);  
 rowsRemaining--;  
 if (rowsRemaining == 0) {  
 enableUI(true);  
 }  
 }  
 }  
  
 public static void main(String[] args)  
 {  
 FractalExplorer displayExplorer = new FractalExplorer(600);  
 displayExplorer.createAndShowGUI();  
 displayExplorer.drawFractal();  
 }  
  
}

**BurningShip.java**

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)  
 {  
 int iteration = 0;  
 double zreal = 0;  
 double zimaginary = 0;  
  
 while (iteration < *MAX\_ITERATIONS* &&  
 zreal \* zreal + zimaginary \* zimaginary < 4)  
 {  
 double zrealUpdated = zreal \* zreal - zimaginary \* zimaginary + x;  
 double zimaginaryUpdated = 2 \* Math.*abs*(zreal)  
 \* Math.*abs*(zimaginary) + y;  
   
 zreal = zrealUpdated;  
 zimaginary = zimaginaryUpdated;  
   
 iteration += 1;  
 }  
  
 if (iteration == *MAX\_ITERATIONS*)  
 {  
 return -1;  
 }  
   
 return iteration;  
 }  
  
 public String toString() {  
 return "Burning Ship";  
 }  
   
}

**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 javax.swing.\*;  
import java.awt.image.\*;  
import java.awt.\*;  
  
class JImageDisplay extends JComponent  
{  
 private BufferedImage displayImage;  
 public BufferedImage getImage() {  
 return displayImage;  
 }  
 public JImageDisplay(int width, int height) {  
 displayImage = new BufferedImage(width, height,  
 BufferedImage.*TYPE\_INT\_RGB*);  
 Dimension imageDimension = new Dimension(width, height);  
 super.setPreferredSize(imageDimension);  
   
 }  
 public void paintComponent(Graphics g)  
 {  
 super.paintComponent(g);  
 g.drawImage(displayImage, 0, 0, displayImage.getWidth(),  
 displayImage.getHeight(), null);  
 }  
 public void clearImage()  
 {  
 int[] blankArray = new int[getWidth() \* getHeight()];  
 displayImage.setRGB(0, 0, getWidth(), getHeight(), blankArray, 0, 1);  
 }  
 public void drawPixel(int x, int y, int rgbColor)  
 {  
 displayImage.setRGB(x, y, rgbColor);  
 }  
}

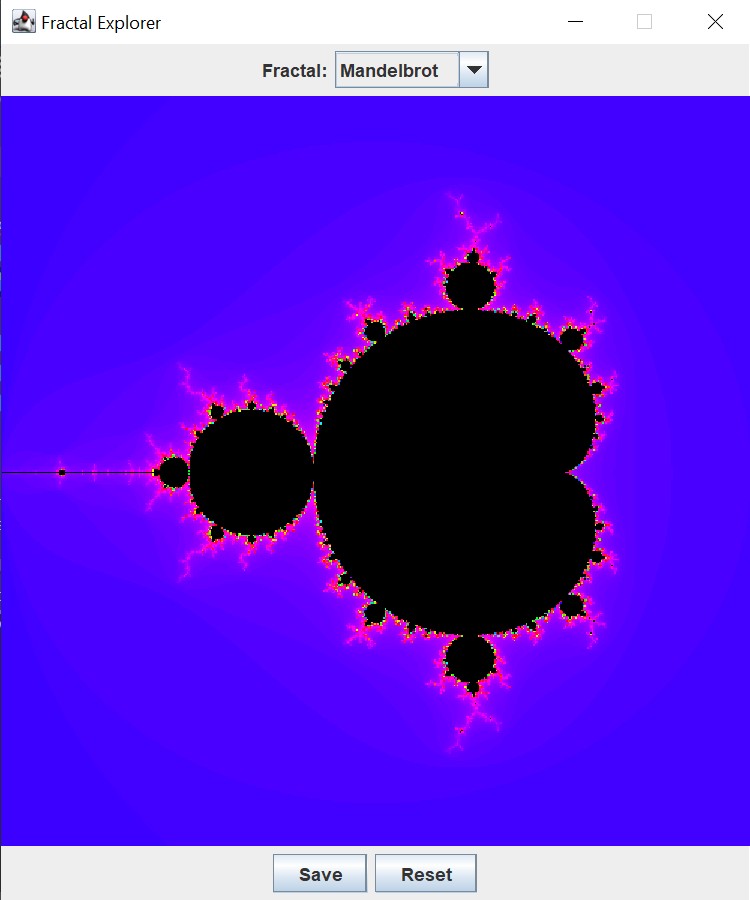
**Mandelbrot.java**

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)  
 {  
 int iteration = 0;  
 double zreal = 0;  
 double zimaginary = 0;  
  
 while (iteration < *MAX\_ITERATIONS* &&  
 zreal \* zreal + zimaginary \* zimaginary < 4)  
 {  
 double zrealUpdated = zreal \* zreal - zimaginary \* zimaginary + x;  
 double zimaginaryUpdated = 2 \* zreal \* zimaginary + y;  
 zreal = zrealUpdated;  
 zimaginary = zimaginaryUpdated;  
 iteration += 1;  
 }  
  
 if (iteration == *MAX\_ITERATIONS*)  
 {  
 return -1;  
 }  
   
 return iteration;  
 }  
  
 public String toString() {  
 return "Mandelbrot";  
 }  
  
}

**Tricorn.java**

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) {  
 int iteration = 0;  
 double zreal = 0;  
 double zimaginary = 0;  
  
 while (iteration < *MAX\_ITERATIONS* &&  
 zreal \* zreal + zimaginary \* zimaginary < 4) {  
 double zrealUpdated = zreal \* zreal - zimaginary \* zimaginary + x;  
 double zimaginaryUpdated = -2 \* zreal \* zimaginary + y;  
 zreal = zrealUpdated;  
 zimaginary = zimaginaryUpdated;  
 iteration += 1;  
 }  
 if (iteration == *MAX\_ITERATIONS*) {  
 return -1;  
 }  
  
 return iteration;  
 }  
 public String toString() {  
 return "Tricorn";  
 }  
}

**Выполнение программы:**



**Вывод:**По результатам выполнения этой ЛР — я научился работать с потоками в Java и улучшил программу рисования фракталов. Благодаря новому подходу — программа больше не зависает при отрисовке фракталов.