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 № 4

«Рисование фракталов»

по дисциплине «Объектно-ориентированное программирование»

Выполнил: студент группы БФИ1901

Кириллов Роман Сергеевич

Проверил: Мосева М.С

Москва, 2020

**Задача:** Создайте класс JImageDisplay, производный от javax.swing.JComponent. Класса должен иметь одно поле с типом доступа private, экземпляп java.awt.image.BufferedImage. Класс BufferedImage управляет изображением, содержимое которого можно записать.

• Конструктор JImageDisplay должен принимать целочисленные значения ширины и высоты, и инициализировать объект BufferedImage новым изображением с этой шириной и высотой, и типом изображения TYPE\_INT\_RGB. Тип определяет, как цвета каждого пикселя будут представлены в изображении; значение TYPE\_INT\_RGB обозначает, что красные, зеленые и синие компоненты имеют по 8 битов, представленные в формате int в указанном порядке. Конструктор также должен вызвать метод setPreferredSize() родительского класса метод с указанной шириной и высотой. (Вы должны будете передать эти значения в объект java.awt.Dimension) Таким образом, когда ваш компонент будет включен в пользовательский интерфейс, он отобразит на экране все изображение.

• Пользовательские компоненты Swing должны предоставлять свой собственный код для отрисовки, переопределяя защищенный метод JComponent paintComponent (Graphics g). Так как наш компонент просто выводит на экран данные изображения, реализация будет очень проста! Во-первых, нужно всегда вызывать метод суперкласса paintComponent (g) так, чтобы объекты отображались правильно. После вызова версии суперкласса, вы можете нарисовать изображение в компоненте, используя следующую операцию: g.drawImage (image, 0, 0, image.getWidth(), image.getHeight(), null); (Мы передаем значение null для ImageObserver, поскольку данная функциональность не требуется.)

• Вы также должны создать два метода с доступом public для записи данных в изображение: метод clearImage (), который устанавливает все пиксели изображения в черный цвет (значение RGB 0), и метод drawPixel (int x, int y, int rgbColor), который устанавливает пиксель в определенный цвет. Оба метода будут необходимы для использования в методе setRGB () класса BufferedImage

написать код для вычисления фрактала Мандельброта. Для создания фракталов используйте следующий исходный файл FractalGenerator.java, от которого будут унаследованы все ваши фрактальные генераторы. Как вы могли заметить данный файл предоставляет также некоторые полезные операции для перевода из экранных координат в систему координат вычисляемого фрактала. Виды фракталов, с которыми нужно будет работать, вычисляются в комплексном виде и включают в себя простые математические функции, которые выполняются многократно, пока не выполнится определенное условие. Функция для фрактала Мандельброта имеет вид: zn = zn-1 2 + c, где все значения — это комплексные числа, z0 = 0, и с - определенная точка фрактала, которую мы отображаем на экране. Вычисления повторяются до тех пор, пока |z| > 2 (в данной ситуации точка находится не во множестве Мандельброта), или пока число итераций не достигнет максимального значения, например, 2000 (в этом случае делается предположение, что точка находится в наборе). Процесс построения фрактала Мандельброта прост: необходимо перебрать все пиксели изображения, рассчитать количество итераций для соответствующей координаты, и затем установить пиксель в цвет, основанный на количестве рассчитанных итераций. Все это будет сделано позже, на данном этапе необходимо реализовать приведенные выше вычисления.

• Создайте подкласс FractalGenerator с именем Mandelbrot. в нем вам необходимо будет обеспечить только два метода: getInitialRange() и numIterations().

• getInitialRange (Rectangle2D.Double) - метод позволяет генератору фракталов определить наиболее «интересную» область комплексной плоскости для конкретного фрактала. Обратите внимание на то, что методу в качестве аргумента передается прямоугольный объект, и метод должен изменить поля прямоугольника для отображения правильного начального диапазона для фрактала. (Пример можно увидеть в методе FractalGenerator.recenterAndZoomRange().) В классе Mandelbrot этот метод должен установить начальный диапазон в (-2 - 1.5i) - (1 + 1.5i). Т.е. значения x и y будут равны -2 и -1.5 соответственно, а ширина и высота будут равны

3. • Метод numIterations(double, double) реализует итеративную функцию для фрактала Мандельброта. Константу с максимальным количеством итераций можно определить следующим образом: public static final int MAX\_ITERATIONS = 2000; Затем вы сможете ссылаться на эту переменную в вашей реализации. Обратите внимание на то, что у Java нет подходящего типа данных для комплексных чисел, поэтому необходимо будет реализовать итеративную функцию, используя отдельные переменные для действительной и мнимой частей. (Вы можете реализовать отдельный класс для комплексных чисел.) Ваш алгоритм должен обладать быстродействием, например, не стоит сравнивать |z| с 2; сравните |z|2 с 22 для того, чтобы избежать сложных и медленных вычислений квадратного корня. Также не стоит использовать метод Math.pow () для вычисления небольших степеней, лучше перемножьте значение, иначе ваш быстродействие вашего кода сильно упадет. В случае, если алгоритм дошел до значения MAX\_ITERATIONS нужно вернуть -1, чтобы показать, что точка не выходит за границы.

Создайте класс FractalExplorer, который позволит вам исследовать различные области фрактала, путем его создания, отображения через графический интерфейс Swing и обработки событий, вызванных взаимодействием приложения с пользователем.

• Класс FractalExplorer должен отслеживать несколько важных полей для состояния программы: 1) Целое число «размер экрана», которое является шириной и высотой отображения в пикселях. (Отображение фрактала будет квадратным.) 2) Ссылка JImageDisplay, для обновления отображения в разных методах в процессе вычисления фрактала. 3) Объект FractalGenerator. Будет использоваться ссылка на базовый класс для отображения других видов фракталов в будущем. 4) Объект Rectangle2D.Double, указывающий диапазона комплексной плоскости, которая выводится на экран. Все вышеприведенные поля будут иметь тип доступа private.

• У класса должен быть конструктор, который принимает значение размера отображения в качестве аргумента, затем сохраняет это значение в соответствующем поле, а также инициализирует объекты диапазона и фрактального генератора. Данный конструктор не должен устанавливать какиелибо компоненты Swing; они будут установлены в следующем методе.

• Создайте метод createAndShowGUI (), который инициализирует графический интерфейс Swing: JFrame, содержащий объект JimageDisplay, и кнопку для сброса отображения. Используйте java.awt.BorderLayout для содержимого окна; добавьте объект отображения изображения в позицию BorderLayout.CENTER и кнопку в позицию BorderLayout.SOUTH. Вам необходимо дать окну подходящий заголовок и обеспечить операцию закрытия окна по умолчанию (см. метод JFrame.setDefaultCloseOperation ()). После того, как компоненты пользовательского интерфейса инициализированы и размещены, добавьте следующую последовательность операций: frame.pack (); frame.setVisible (true); frame.setResizable (false); Данные операции правильно разметят содержимое окна, сделают его видимым (окна первоначально не отображаются при их создании для того, чтобы можно было сконфигурировать их прежде, чем выводить на экран), и затем запретят изменение размеров окна.

• Реализуйте вспомогательный метод с типом доступа private для вывода на экран фрактала, можете дать ему имя drawFractal (). Этот метод должен циклически проходить через каждый пиксель в отображении (т.е. значения x и y будут меняться от 0 до размера отображения), и сделайте следующее: ¬ Вычислите количество итераций для соответствующих координат в области отображения фрактала. Вы можете определить координаты с плавающей точкой для определенного набора координат пикселей, используя вспомогательный метод FractalGenerator.getCoord (); например, чтобы получить координату x, соответствующую координате пикселя X, сделайте следующее: //x - пиксельная координата; xCoord - координата в пространстве фрактала double xCoord = FractalGenerator.getCoord (range.x, range.x + range.width, displaySize, x); ¬ Если число итераций равно -1 (т.е. точка не выходит за границы, установите пиксель в черный цвет (для rgb значение 0). Иначе выберите значение цвета, основанное на количестве итераций. Можно также для этого использовать цветовое пространство HSV: поскольку значение цвета варьируется от 0 до 1, получается плавная последовательность цветов от красного к желтому, зеленому, синему, фиолетовому и затем обратно к красному! Для этого вы можете использовать следующий фрагмент: float hue = 0.7f + (float) numIters / 200f; int rgbColor = Color.HSBtoRGB(hue, 1f, 1f); Если вы придумали другой способ отображения пикселей в зависимости от количества итераций, попробуйте реализовать его! ¬ Отображение необходимо обновлять в соответствии с цветом для каждого пикселя. ¬ После того, как вы закончили отрисовывать все пиксели, вам необходимо обновить JimageDisplay в соответствии с текущим изображением. Для этого вызовите функцию repaint() для компонента. В случае, если вы не воспользуетесь данным методом, изображение на экране не будет обновляться!

• Создайте внутренний класс для обработки событий java.awt.event.ActionListener от кнопки сброса. Обработчик должен сбросить диапазон к начальному, определенному генератором, а затем перерисовать фрактал. После того, как вы создали этот класс, обновите метод createAndShowGUI ().

• Создайте другой внутренний класс для обработки событий java.awt.event.MouseListener с дисплея. Вам необходимо обработать события от мыши, поэтому вы должны унаследовать этот внутренний класс от класса MouseAdapterAWT. При получении события о щелчке мышью, класс должен отобразить пиксельные кооринаты щелчка в область фрактала, а затем вызвать метод генератора recenterAndZoomRange() с координатами, по которым щелкнули, и масштабом 0.5. Таким образом, нажимая на какое-либо место на фрактальном отображении, вы увеличиваете его! Не забывайте перерисовывать фрактал после того, как вы меняете область фрактала. Далее обновите метод createAndShowGUI (), чтобы зарегистрировать экземпляр этого обработчика в компоненте фрактального отображения.

• В заключении, вам необходимо создать статический метод main() для FractalExplorer так, чтобы можно было его запустить. В main необходимо будет сделать:

¬ Инициализировать новый экземпляр класса FractalExplorer с размером отображения 800.

¬ Вызовите метод createAndShowGUI () класса FractalExplorer.

¬ Вызовите метод drawFractal() класса FractalExplorer для отображения начального представления.

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 = x \* x - y \* y + r;  
// i1 = 2 \* x \* y + i;  
// r = r1;  
// i = i1;  
// k++;  
// }  
 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;  
 }  
}

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 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.swing.JButton;  
import javax.swing.JFrame;  
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;  
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);  
 }  
 public void createAndShowGUI() {  
 JDisplay.setLayout(new BorderLayout());  
 JFrame frame = new JFrame("Fractal Explorer");  
 frame.add(JDisplay, BorderLayout.*CENTER*);  
 JButton resetButton = new JButton("Reset Display");  
 reset rez = new reset();  
 resetButton.addActionListener((ActionListener) rez);  
 frame.add(resetButton, BorderLayout.*SOUTH*);  
 Clicker klik = new Clicker();  
 JDisplay.addMouseListener(klik);  
 frame.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 frame.pack();  
 frame.setVisible(true);  
 frame.setResizable(false);  
 }  
 private void drawFractal() {  
 double x1, y1;  
 for (int x = 0; x < si; x++) {  
 for (int y = 0; y < si; y++) {  
 x1 = FractalGenerator.*getCoord*(d.x, d.x + d.width, si, x);  
 y1 = FractalGenerator.*getCoord*(d.y, d.y + d.height, si, y);  
 int k = gen.numIterations(x1, y1);  
 if (k == -1) {  
 JDisplay.drawPixel(x, y, 0);  
 } else {  
 float hue = 0.7f + (float) k / 200f;  
 int rgbColor = Color.*HSBtoRGB*(hue, 1f, 1f);  
 JDisplay.drawPixel(x, y, rgbColor);  
 }  
 }  
 }  
 JDisplay.repaint();  
 }  
 private class reset implements ActionListener  
 {  
 public void actionPerformed(ActionEvent e)  
 {  
 gen.getInitialRange(d);  
 drawFractal();  
 }  
 }  
 private class Clicker extends MouseAdapter  
 {  
 @Override  
 public void mouseClicked(MouseEvent e) {  
 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();  
 }  
}

Результат работы:

