**Лабораторная работа № 2**

**«Объектная модель языка Java»**

**Цель работы:** изучение объектной модели Java, классы, интерфейсы, наследование, полиморфизм.

**Задание на лабораторную работу**

В ходе выполнения данной лабораторной работы необходимо ознакомиться с понятиями классы, полиморфизм, наследование и интерфейсы. Разработать диаграмму классов, согласно заданному варианту. Создать приложение, демонстрирующее работоспособность модели.

**Краткие теоретические сведения**

Элементарным строительным блоком в Java является класс. Класс–это конструкция, определяющая поля данных объектов данного класса и их поведение. Практически класс в Java сам по себе не является объектом. Это лишь шаблон, который определяет, из каких частей будет состоять объект, созданный с помощью этого класса, и как он будет себя вести.

В Java кроме обычных классов существуют и абстрактные классы. Абстрактный класс – это класс, который не может иметь экземпляров. Любой класс, который имеет хотя бы один абстрактный метод или поле должен быть описан как абстрактный. С помощью ключевого слова ***abstract*** можно указать что класс, метод или поле являются абстрактными. В качестве примера создадим абстрактный класс Shape, который будет характеризовать любую геометрическую фигуру.

public abstract class Shape implements IShape

{

private Point startPoint;

private Color color;

public Shape(int x, int y)

{

this.startPoint = new Point(x, y);

}

public Point getStartPoint()

{

return startPoint;

}

public void setStartPoint(Point center)

{

this.startPoint = center;

}

public Color getColor()

{

return color;

}

public void setColor(Color color)

{

this.color = color;

}

}

В данном абстрактном классе описаны два поля – начальная точка на плоскости, которая определяет положение фигуры в пространстве и цвет. Также созданы методы для доступа к этим полям.

Как видно из описания, класс Shape реализует интерфейс IShape. Под реализацией подразумевается описание конкретных действий, которые должен выполнять метод, описанный в интерфейсе. Интерфейсы, которые реализует класс, указываются после ключевого слова ***implements***. Интерфейс – это упрощенное описание класса, в его описании указывают только сигнатуру методов без указания конкретной реализации.

Интерфейс IShape выглядит так:

public interface IShape

{

public void draw(Graphics canvas);

}

Здесь описана сигнатура одного единственного метода, который принимает в качестве параметра объект класса ***Graphics*** (графический контекст любого компонента). При помощи технологии Java 2D можно производить рисование графических примитивов на контексте. Техника работы с Java 2D будет рассмотрена ниже.

Таким образом, описав абстрактный класс Shape, реализующий интерфейс IShape получают базовую конструкцию для создания класса любой геометрической фигуры.

Далее создаются три класса, описывающие различные геометрические фигуры – круг, треугольник и прямоугольник:

public class Circle extends Shape

{

public Circle()

{

this(0, 0);

}

public Circle(int x, int y)

{

super(x, y);

}

public void draw(Graphics canvas)

{

canvas.setColor(getColor());

canvas.fillOval(getStartPoint().x, getStartPoint().y, 50, 50);

}

}

public class Triangle extends Shape

{

public Triangle()

{

this(0, 0);

}

public Triangle(int x, int y)

{

super(x, y);

}

public void draw(Graphics canvas)

{

canvas.setColor(getColor());

Polygon p = new Polygon();

p.addPoint(getStartPoint().x, getStartPoint().y);

p.addPoint(getStartPoint().x + 40, getStartPoint().y);

p.addPoint(getStartPoint().x, getStartPoint().y + 30);

canvas.fillPolygon(p);

}

}

public class Rectangle extends Shape

{

public Rectangle()

{

this(0, 0);

}

public Rectangle(int x, int y)

{

super(x, y);

}

public void draw(Graphics canvas)

{

canvas.setColor(getColor());

canvas.fillRect(getStartPoint().x,

getStartPoint().y, 50, 50);

}

}

Все описанные классы имеют одного родителя – абстрактный класс Shape. Данный подход позволяет созданным классам унаследовать все свойства родительского класса, такие как цвет и начальная точка на плоскости, и указать конкретную реализацию метода ***draw***.

Рассмотрим метод рисования фигуры подробнее. В качестве входного параметра он принимает графический контекст, на котором в последующем можно будет нарисовать ту или иную фигуру. Перед тем как начать отображать геометрическую фигуру на контексте, необходимо установить текущий цвет контекста (по умолчанию он черный). С помощью метода ***setColor(Color c)*** производятся эти действия. В нашем случае цветом будет служить тот цвет, который был задан фигуре при инициализации класса.

В наборе технологии Java 2D существует несколько подходов по рисованию графических примитивов.

Первый – это создание объекта класса ***Polygon*** и с помощью метода ***addPoint(int x, int y)*** заполнение его точками. Затем полигон передается в качестве входного параметра методу ***fillPolygon(Polygon p)***, который произведет его отрисовку и заливку текущим цветом контекста. Данный подход используется для отображения треугольника.

Второй – это отрисовка простых геометрических фигур (окружность, овал, прямоугольник…) при помощи стандартных методов графического ко текста ***fillOval(int x, int y, int width, int height)*** и ***fillRect(int x, int y, int width,*** ***int height)***. Данный подход используется для отображения круга и прямоугольника.

На данный момент создано три класса, описывающие простейшие геометрические фигуры, с реализацией методов их отрисовки на графическом контексте любого компонента визуального интерфейса.

Диаграмма, описывающая иерархическую структуру классов, изображена на рисунке 1.

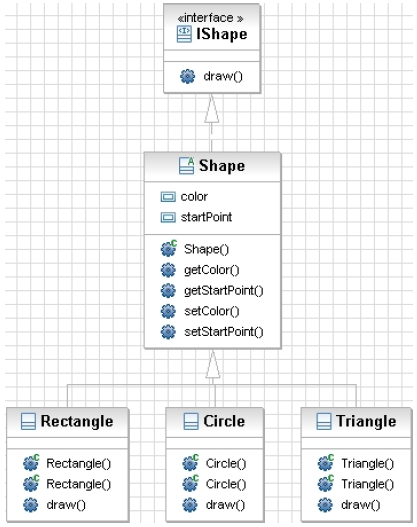


Рисунок 1 – Диаграмма классов тестового примера

Диаграммы классов отображают различные классы, из которых состоит система, и их взаимодействие между собой. Диаграммы классов являются «статичными» диаграммами, потому что они отображают классы вместе с их методами и атрибутами, так же как и статические взаимодействия между ними: какие классы «знают» о каких классах или какие классы «являются частью» других классов, но не отображают обмен сообщениями между классами.

Класс определяет атрибуты и методы набора объектов. Все объекты класса (называемые экземплярами) имеют одинаковое поведение и одинаковый набор атрибутов (у каждого объекта - собственный набор атрибутов). Иногда вместо класса используется термин «тип», но важно понимать, что эти термины неодинаковы. Термин «Тип» имеет более общий смысл.

В UML классы представлены прямоугольниками с именем класса, которые могут отображать атрибуты и операции класса, помещённые внутри прямоугольника.

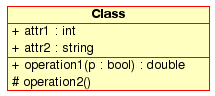


Рисунок 2 - Визуальное представление класса в UML

В UML атрибуты отображаются как минимум их названием, но также могут показывать свой тип, начальное значение и другие свойства. Атрибуты также могут отображаться с указанием их области видимости:

* + отмечает *public* атрибуты
* # отмечает *protected* атрибуты
* - отмечает *private* атрибуты

Методы (операции) также отображаются как минимум их названием, но также могут показывать свои параметры и возвращаемый тип. Подобно атрибутам, методы имеют область видимости:

* + отмечает *public* операции;
* # отмечает *protected* операции;
* - отмечает *private* операции.

Классы могут взаимодействовать друг с другом различными способами.

1. **Обобщение**

Наследование является основной концепцией объектно-ориентированного программирования, в которой класс «получает» все атрибуты и методы наследуемого класса и может переопределять/модифицировать некоторые из них, имея возможность добавлять собственные атрибуты и методы.

В UML ассоциация *обобщения* между двумя классами помещает их в иерархию, представляющую концепцию наследования производного класса от базового. В UML обобщения отображаются в виде линии, соединяющей два класса, со стрелочкой у базового класса.

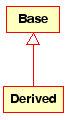


Рисунок 3 - Визуальное представление обобщения в UML

1. **Ассоциации**

Ассоциации отображают взаимодействия между классами и определяют общую семантику и структуру многих типов «связей» между объектами.

Ассоциации являются механизмом, который позволяет объектам взаимодействовать друг с другом. Они описывают связи между различными классами (связи между объектами называются соединениями объектов или *ссылками*).

Ассоциации могут иметь роль, которая определяет назначение ассоциации и может быть одно- или двунаправленной, показывая, могут ли взаимодействующие объекты обмениваться сообщениями, или только один объект может отсылать сообщения. Каждый конец ассоциации имеет описание множественности, которое указывает, какое количество объектов на этой стороне ассоциации может взаимодействовать с одним объектом на другой стороне.

В UML ассоциации отображаются в виде линии, соединяющей взаимодействующие классы, и могут показывать роль и количество объектов с каждой стороны. Множественность отображается в виде диапазона [мин..макс] неотрицательных значений, звёздочка (\*) в качестве максимального значения обозначает бесконечность.

Визуальное представление ассоциации в UML

Рисунок 4 - Визуальное представление ассоциации в UML

1. **Объединение**

Объединения являются специальным типом ассоциаций, в котором два участвующих класса не равнозначны по статусу, но создают взаимодействие типа «часть-целое». Объединение описывает, как класс, играющий роль целого, создаётся из других классов, играющих роль частей. Для объединений класс, выступающий в роли целого, всегда имеет множественность, равную единице.

В UML объединения отображаются в виде линии с ромбом на стороне целого, соединяющей взаимодействующие классы.

Визуальное представление объединения в UML

Рисунок 5 - Визуальное представление объединения в UML

1. **Композиция**

Композиция является ассоциацией, которая представляет *очень тесное* объединение. Это означает, что композиция формирует взаимодействия типа «часть-целое», но эти взаимодействия настолько сильные, что части не могут существовать сами по себе. Они существуют только внутри целого, и при уничтожении целого уничтожаются и части.

В UML композиции отображаются в виде линии с закрашенным ромбом на стороне целого, соединяющей взаимодействующие классы.

Визуальное представление композиции в UML

Рисунок 6 - Визуальное представление композиции в UML

1. **Другие элементы диаграммы класса**

Диаграммы классов могут содержать другие элементы внутри классов.

**Интерфейсы.** Интерфейсы являются абстрактными классами, следовательно, объекты данных классов не могут быть созданы напрямую. Они могут содержать методы, но не атрибуты. Классы могут наследоваться от интерфейсов (через ассоциацию реализации), и полученные объекты затем могут использоваться при составлении диаграмм. Для изображения интерфейса используется специальный графический символ - прямоугольник класса с ключевым словом или стереотипом «interface». При этом секция атрибутов у прямоугольника отсутствует, а указывается только секция операций.

**Типы данных.** Типы данных являются примитивами, которые обычно встроены в язык программирования. Яркими примерами могут служить целые и логические типы. Они могут не иметь отношения к классам, но классы могут использовать их.

**Перечисления.** Перечисления представляют собой простой список значений. Примером перечисления является перечисление дней недели. Подобно типам данных они могут использоваться в классах.

**Пакеты.** Пакеты представляют собой пространства имён в языке программирования. В диаграмме они используются для представления частей системы, которые содержат более одного класса (число классов может превышать несколько сотен).

Для работы с UML имеются различные программные средства, например, IBM Rational Rose, Borland Together, Microsoft Visio, Sparx Systems Enterprise Architect, Gentleware Poseidon, SmartDraw, Dia, Telelogic TAU G2, StarUML и другие. В рамках данной лабораторной работы можно воспользоваться возможностями Microsoft Visio или установить **подключаемый модуль UML для среды IDE NetBeans. К сожалению, подключаемый модуль UML для среды IDE NetBeans доступен только для версии 6.7 и более ранних версий. Для более поздних версий NetBeans рекомендуется некоммерческая версия SDE для NetBeans от Visual Paradigm (**<http://netbeans.org/features/uml/index_ru.html>**).**

Для тестирования созданных классов создается новый класс ***Main.***

public class Main extends JComponent

{

public static void main(String[] args)

{

JFrame frame = new JFrame();

frame.setSize(400, 400);

frame.setTitle("Shape Example");

frame.getContentPane().add(new Main());

frame.setVisible(true);

}

public void paint(Graphics g)

{

Shape shape = new Rectangle(10, 10);

shape.setColor(Color.RED);

shape.draw(g);

shape = new Circle(100, 100);

shape.setColor(Color.BLUE);

shape.draw(g);

shape = new Triangle(200,200);

shape.setColor(Color.BLACK);

shape.draw(g);

}

}

Пример работы класса ***Main*** показан на рисунке 2.

****

Рисунок 3.2 – Пример работы тестового проекта

**Порядок выполнения работы**

1. Ознакомиться с заданием на лабораторную работу
2. Разработать упрощенную иерархическую модель классов предметной области согласно варианту задания в таблице 1 (столбец 2)
3. Изучить программную реализацию одного из фрагментов исследуемой предметной области (столбец 3). Код программы приведен в приложении к данной лабораторной работе
4. Реорганизовать код в соответствии с разработанной диаграммой классов
5. Проверить работоспособность реорганизованного программного кода
6. Составить отчет о выполнении лабораторной работы

Таблица 1 – Варианты заданий

|  |  |  |
| --- | --- | --- |
| Вариант | Предметная область | Программная реализация |
| 1 | Фрактальные кластеры | Фрактал DLA (Diffusion Limited Aggregation) |
| 2 | Фрактальные поверхности | Фрактал с алгоритмом срединного смещения |
| 3 | Стохастические фракталы | Рандомизированные снежинки Коха |
| 4 | Геометрические фракталы | Снежинки Коха |
| 5 | Детерминированные фракталы | Цветные снежинки Коха |
| 6 | Регулярные фракталы | Цветные антиснежинки Коха |
| 7 | Самоаффинные фракталы | Цветные мегаснежинки Коха |
| 8 | Фрактальные деревья и графы | Дерево Пифагора |

**Содержание отчета**

1. Тема лабораторной работы
2. Цель лабораторной работы
3. Описание и диаграмма классов предметной области
4. Разработка математического и алгоритмического обеспечения
5. Программная реализация фрагмента предметной области
6. Выводы

**Приложение**

***Фрактал DLA***

package fractal;

import java.awt.\*;

import java.awt.geom.\*;

import java.awt.image.\*;

import java.io.\*;

import java.util.Random;

import javax.swing.\*;

public class Claster

{

static final int WIDTH = 512;

static final int HEIGHT = 512;

static final int COUNT = 100000;

static BufferedImage image;

static Graphics2D graphics;

private static void drawDLA()

{

Random rnd = new Random();

for (int i = 0; i < COUNT; ++i)

{

int x = WIDTH / 2;

int y = HEIGHT / 2;

int dx = 1;

int dy = 1;

while (image.getRGB(x, y) != Color.BLACK.getRGB())

{

dx = rnd.nextInt(3) - 1;

dy = rnd.nextInt(3) - 1;

x += dx;

y += dy;

}

image.setRGB(x - dx, y - dy, Color.BLACK.getRGB());

System.out.println(i);

}

}

public static void main(String[] args) throws IOException

{

image = new BufferedImage(WIDTH, HEIGHT, BufferedImage.TYPE\_INT\_RGB);

graphics = image.createGraphics();

graphics.setColor(Color.WHITE);

graphics.fill(new Rectangle2D.Double(0, 0, WIDTH, HEIGHT));

graphics.setColor(Color.BLACK);

graphics.drawRect(1, 1, WIDTH - 2, HEIGHT - 2);

drawDLA();

JFrame frame = new JFrame();

frame.addNotify();

frame.setSize(frame.getInsets().left

+ frame.getInsets().right + WIDTH,

frame.getInsets().top

+ frame.getInsets().bottom + HEIGHT);

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.add(new JPanel()

{

@Override

public void paintComponent(Graphics g) {

Graphics2D g2 = (Graphics2D) g;

if (image != null) {

g2.drawImage(image, 0, 0, null);

}

}

});

frame.setVisible(true);

}

}

***Фрактал с алгоритмом срединного смещения***

package fractal;

import java.awt.Color;

import java.awt.Graphics;

import java.awt.Graphics2D;

import java.awt.Image;

import java.awt.image.BufferedImage;

import javax.swing.JFrame;

import javax.swing.JPanel;

public class Surface

{

static float[][] x;

static void calc(int a, int b, int c, int d)

{

if (a + 1 == c || b + 1 == d)

{

return;

}

float r = (c - a) + (d - b);

int e = (a + c) / 2;

int f = (b + d) / 2;

x[e][f] =

(x[a][b] + x[a][d] + x[c][b] + x[c][d]) / 4 +

r \* (float)(Math.random() - .5);

x[e][b] = (x[a][b] + x[c][b]) / 2;

x[e][d] = (x[a][d] + x[c][d]) / 2;

x[a][f] = (x[a][b] + x[a][d]) / 2;

x[c][f] = (x[c][b] + x[c][d]) / 2;

calc(a, b, e, f);

calc(a, f, e, d);

calc(e, b, c, f);

calc(e, f, c, d);

}

static Color floatToColor(float x)

{

if (x <= 0)

{

return new Color(1f, 0f, 0f);

}

if (x <= 1f/6)

{

return new Color(1f, x \* 6f, 0f);

}

if (x <= 2f/6)

{

return new Color(1f - (x - 1f/6) \* 6f, 1f, 0f);

}

if (x <= 3f/6)

{

return new Color(0f, 1f, (x - 2f/6f) \* 6f);

}

if (x <= 4f/6)

{

return new Color(0f, 1f - (x - 3f/6f) \* 6f, 1f);

}

if (x <= 5f/6)

{

return new Color((x - 4f/6) \* 6f, 0f, 1f);

}

if (x < 1f)

{

return new Color(1f, 0f, 1f - (x - 5f/6f) \* 6f);

}

return new Color(1f, 0f, 0f);

}

static BufferedImage createPlasma(int countLevels)

{

int size = (int)Math.round(Math.pow(2, countLevels));

BufferedImage image = new BufferedImage(size + 1, size + 1,

BufferedImage.TYPE\_INT\_RGB);

x = new float[size + 1][size + 1];

x[0][0] = (float)Math.random() \* size / 2 - size / 4;

x[0][size] = (float)Math.random() \* size / 2 - size / 4;

x[size][0] = (float)Math.random() \* size / 2 - size / 4;

x[size][size] = (float)Math.random() \* size / 2 - size / 4;

calc(0, 0, size, size);

float min = Float.MAX\_VALUE;

float max = Float.MIN\_VALUE;

for (int i = 0; i <= size; ++i)

{

for (int j = 0; j <= size; ++j)

{

if (x[i][j] < min)

{

min = x[i][j];

}

if (x[i][j] > max)

{

max = x[i][j];

}

}

}

for (int i = 0; i <= size; ++i)

{

for (int j = 0; j <= size; ++j)

{

image.setRGB(j, i,

floatToColor((x[i][j] - min) / (max - min)).getRGB());

}

}

return image;

}

static Image image;

public static void main(String[] args)

{

image = createPlasma(9);

JFrame frame = new JFrame();

frame.addNotify();

frame.setSize(frame.getInsets().left

+ frame.getInsets().right + image.getWidth(null),

frame.getInsets().top

+ frame.getInsets().bottom + image.getHeight(null));

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.add(new JPanel()

{

@Override

public void paintComponent(Graphics g) {

Graphics2D G = (Graphics2D) g;

if (image != null) {

G.drawImage(image, 0, 0, null);

}

}

});

frame.setVisible(true);

}

}

***Рандомизированные снежинки Коха***

package fractal;

import java.awt.Color;

import java.awt.Graphics;

import java.awt.Graphics2D;

import java.awt.geom.Path2D;

import java.awt.geom.Point2D;

import java.awt.geom.Rectangle2D;

import java.awt.image.BufferedImage;

import java.util.Random;

import javax.swing.JFrame;

import javax.swing.JPanel;

public class RandomKoch

{

private static final int WIDTH = 640;

private static final int HEIGHT = 480;

private static BufferedImage image;

private static Graphics2D graph;

private static Random random = new Random();

private static int randomSgn()

{

return random.nextInt(2) \* 2 - 1;

}

private static void drawRandomKochCurve(Point2D p, Point2D q, int n,

Color inColor, Color outColor)

{

int w = randomSgn();

Point2D r = new Point2D.Double(

(2 \* p.getX() + q.getX()) / 3,

(2 \* p.getY() + q.getY()) / 3);

Point2D s = new Point2D.Double(

(p.getX() + q.getX()) / 2 -

w \* (p.getY() - q.getY()) \* Math.sqrt(3) / 6,

((p.getY() + q.getY()) / 2 +

w \* (p.getX() - q.getX()) \* Math.sqrt(3) / 6));

Point2D t = new Point2D.Double(

(p.getX() + 2 \* q.getX()) / 3,

(p.getY() + 2 \* q.getY()) / 3);

Path2D path = new Path2D.Double();

path.moveTo(r.getX(), r.getY());

path.lineTo(s.getX(), s.getY());

path.lineTo(t.getX(), t.getY());

path.lineTo(r.getX(), r.getY());

path.closePath();

if (w == 1)

{

graph.setColor(inColor);

}

else /\* if w == -1 \*/

{

graph.setColor(outColor);

}

graph.fill(path);

if (n == 0)

{

return;

}

drawRandomKochCurve(p, r, n - 1, inColor, outColor);

drawRandomKochCurve(r, s, n - 1, inColor, outColor);

drawRandomKochCurve(s, t, n - 1, inColor, outColor);

drawRandomKochCurve(t, q, n - 1, inColor, outColor);

}

private static void drawRandomKochSnowflake(Point2D c, double d,

int m, int n,

Color inColor, Color outColor)

{

Point2D[] vs = new Point2D[m];

for (int i = 0; i < m; ++i)

{

vs[i] = new Point2D.Double(

c.getX() + d \* Math.cos(2 \* Math.PI / m \* i),

c.getY() - d \* Math.sin(2 \* Math.PI / m \* i));

}

Path2D path = new Path2D.Double();

path.moveTo(vs[0].getX(), vs[0].getY());

for (int i = 0; i < m; ++i)

{

path.lineTo(vs[(i + 1) % m].getX(), vs[(i + 1) % m].getY());

}

path.closePath();

graph.setColor(inColor);

graph.fill(path);

for (int i = 0; i < m; ++i)

{

drawRandomKochCurve(vs[(i + 1) % m], vs[i], n, inColor, outColor);

}

}

private static Color getBetweenColor(

Color startColor, Color endColor, double p)

{

return new Color(

(int)(startColor.getRed() +

(endColor.getRed() - startColor.getRed()) \* p),

(int) (startColor.getGreen() +

(endColor.getGreen() - startColor.getGreen()) \* p),

(int) (startColor.getBlue() +

(endColor.getBlue() - startColor.getBlue()) \* p));

}

private static void drawKochMegaSnowflake(Point2D c, double d,

int m, int k, int n,

Color inColor, Color endColor, Color outColor)

{

drawRandomKochSnowflake(c, d, m, n, endColor, outColor);

for (int i = 1; i < k; ++i)

{

drawRandomKochSnowflake(c, d \* (k - i) / k, m, n,

getBetweenColor(endColor, inColor, (double) i / k),

getBetweenColor(endColor, inColor, (double) (i - 1) / k));

}

}

public static void main(String[] args)

{

image = new BufferedImage(WIDTH, HEIGHT, BufferedImage.TYPE\_INT\_RGB);

graph = image.createGraphics();

graph.setColor(Color.BLUE);

graph.fill(new Rectangle2D.Double(0, 0, WIDTH, HEIGHT));

// drawRandomKochSnowflake(new Point2D.Double(WIDTH / 2, HEIGHT / 2),

// WIDTH / 3, 4, 4, Color.GREEN, Color.BLUE);

drawKochMegaSnowflake(new Point2D.Double(WIDTH / 2, HEIGHT / 2),

WIDTH / 3, 6, 3, 4, Color.GREEN, Color.ORANGE, Color.BLUE);

JFrame frame = new JFrame();

frame.addNotify();

frame.setSize(frame.getInsets().left +

frame.getInsets().right + WIDTH,

frame.getInsets().top +

frame.getInsets().bottom + HEIGHT);

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.add(new JPanel()

{

@Override

public void paintComponent(Graphics g) {

Graphics2D G = (Graphics2D) g;

if (image != null) {

G.drawImage(image, 0, 0, null);

}

}

});

frame.setVisible(true);

}

}

***Снежинки Коха***

package fractal;

import java.awt.Color;

import java.awt.Graphics;

import java.awt.Graphics2D;

import java.awt.geom.Line2D;

import java.awt.geom.Point2D;

import java.awt.geom.Rectangle2D;

import java.awt.image.BufferedImage;

import javax.swing.JFrame;

import javax.swing.JPanel;

public class KochSnowflakes

{

private static final int WIDTH = 640;

private static final int HEIGHT = 480;

private static BufferedImage image;

private static Graphics2D graph;

private static void drawKochCurve(Point2D p, Point2D q, int n)

{

if (n == 0)

{

graph.draw(new Line2D.Double(p, q));

return;

}

Point2D r = new Point2D.Double(

(2 \* p.getX() + q.getX()) / 3,

(2 \* p.getY() + q.getY()) / 3);

Point2D s = new Point2D.Double(

(p.getX() + q.getX()) / 2 -

(p.getY() - q.getY()) \* Math.sqrt(3) / 6,

((p.getY() + q.getY()) / 2 +

(p.getX() - q.getX()) \* Math.sqrt(3) / 6));

Point2D t = new Point2D.Double(

(p.getX() + 2 \* q.getX()) / 3,

(p.getY() + 2 \* q.getY()) / 3);

drawKochCurve(p, r, n - 1);

drawKochCurve(r, s, n - 1);

drawKochCurve(s, t, n - 1);

drawKochCurve(t, q, n - 1);

}

private static void drawKochSnowflake(Point2D c, double d, int m, int n)

{

Point2D[] vs = new Point2D[m];

for (int i = 0; i < m; ++i)

{

vs[i] = new Point2D.Double(

c.getX() + d \* Math.cos(2 \* Math.PI / m \* i),

c.getY() - d \* Math.sin(2 \* Math.PI / m \* i));

}

for (int i = 0; i < m; ++i)

{

drawKochCurve(vs[(i + 1) % m], vs[i], n);

}

}

public static void main(String[] args)

{

image = new BufferedImage(WIDTH, HEIGHT, BufferedImage.TYPE\_INT\_RGB);

graph = image.createGraphics();

graph.setColor(Color.WHITE);

graph.fill(new Rectangle2D.Double(0, 0, WIDTH, HEIGHT));

graph.setColor(Color.BLACK);

drawKochSnowflake(new Point2D.Double(WIDTH / 2, HEIGHT / 2),

WIDTH / 3, 2, 4);

JFrame frame = new JFrame();

frame.addNotify();

frame.setSize(frame.getInsets().left +

frame.getInsets().right + WIDTH,

frame.getInsets().top +

frame.getInsets().bottom + HEIGHT);

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.add(new JPanel()

{

@Override

public void paintComponent(Graphics g)

{

Graphics2D G = (Graphics2D) g;

if (image != null)

{

G.drawImage(image, 0, 0, null);

}

}

});

frame.setVisible(true);

}

}

***Цветные снежинки Коха***

package fractal;

import java.awt.Color;

import java.awt.Graphics;

import java.awt.Graphics2D;

import java.awt.geom.Line2D;

import java.awt.geom.Point2D;

import java.awt.geom.Rectangle2D;

import java.awt.image.BufferedImage;

import java.awt.geom.Path2D;

import javax.swing.JFrame;

import javax.swing.JPanel;

public class ColorKochSnowflakes

{

private static final int WIDTH = 640;

private static final int HEIGHT = 480;

private static BufferedImage image;

private static Graphics2D graph;

private static void drawKochCurve(Point2D p, Point2D q, int n, Color color)

{

if (n == 0)

{

graph.setColor(Color.BLACK);

graph.draw(new Line2D.Double(p, q));

return;

}

Point2D r = new Point2D.Double(

(2 \* p.getX() + q.getX()) / 3,

(2 \* p.getY() + q.getY()) / 3);

Point2D s = new Point2D.Double(

(p.getX() + q.getX()) / 2 -

(p.getY() - q.getY()) \* Math.sqrt(3) / 6,

((p.getY() + q.getY()) / 2 +

(p.getX() - q.getX()) \* Math.sqrt(3) / 6));

Point2D t = new Point2D.Double(

(p.getX() + 2 \* q.getX()) / 3,

(p.getY() + 2 \* q.getY()) / 3);

Path2D path = new Path2D.Double();

path.moveTo(r.getX(), r.getY());

path.lineTo(s.getX(), s.getY());

path.lineTo(t.getX(), t.getY());

path.lineTo(r.getX(), r.getY());

path.closePath();

graph.setColor(color);

graph.fill(path);

drawKochCurve(p, r, n - 1, color);

drawKochCurve(r, s, n - 1, color);

drawKochCurve(s, t, n - 1, color);

drawKochCurve(t, q, n - 1, color);

}

private static void drawKochSnowflake(Point2D c, double d,

int m, int n, Color color)

{

Point2D[] vs = new Point2D[m];

for (int i = 0; i < m; ++i) {

vs[i] = new Point2D.Double(

c.getX() + d \* Math.cos(2 \* Math.PI / m \* i),

c.getY() - d \* Math.sin(2 \* Math.PI / m \* i));

}

Path2D path = new Path2D.Double();

path.moveTo(vs[0].getX(), vs[0].getY());

for (int i = 0; i < m; ++i) {

path.lineTo(vs[(i + 1) % m].getX(), vs[(i + 1) % m].getY());

}

path.closePath();

graph.setColor(color);

graph.fill(path);

for (int i = 0; i < m; ++i) {

drawKochCurve(vs[(i + 1) % m], vs[i], n, color);

}

}

public static void main(String[] args)

{

image = new BufferedImage(WIDTH, HEIGHT, BufferedImage.TYPE\_INT\_RGB);

graph = image.createGraphics();

graph.setColor(Color.WHITE);

graph.fill(new Rectangle2D.Double(0, 0, WIDTH, HEIGHT));

graph.setColor(Color.BLACK);

drawKochSnowflake(new Point2D.Double(WIDTH / 2, HEIGHT / 2),

WIDTH / 3, 2, 4, Color.GREEN);

JFrame frame = new JFrame();

frame.addNotify();

frame.setSize(frame.getInsets().left +

frame.getInsets().right + WIDTH,

frame.getInsets().top +

frame.getInsets().bottom + HEIGHT);

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.add(new JPanel()

{

@Override

public void paintComponent(Graphics g)

{

Graphics2D G = (Graphics2D) g;

if (image != null)

{

G.drawImage(image, 0, 0, null);

}

}

});

frame.setVisible(true);

}

}

***Цветные антиснежинки Коха***

package fractal;

import java.awt.Color;

import java.awt.Graphics;

import java.awt.Graphics2D;

import java.awt.geom.Line2D;

import java.awt.geom.Point2D;

import java.awt.geom.Rectangle2D;

import java.awt.image.BufferedImage;

import java.awt.geom.Path2D;

import javax.swing.JFrame;

import javax.swing.JPanel;

public class KochAntiSnowflakes

{

private static final int WIDTH = 640;

private static final int HEIGHT = 480;

private static BufferedImage image;

private static Graphics2D graph;

private static void drawKochCurve(Point2D p, Point2D q, int n, Color color)

{

if (n == 0)

{

graph.setColor(Color.BLACK);

graph.draw(new Line2D.Double(p, q));

return;

}

Point2D r = new Point2D.Double(

(2 \* p.getX() + q.getX()) / 3,

(2 \* p.getY() + q.getY()) / 3);

Point2D s = new Point2D.Double(

(p.getX() + q.getX()) / 2 -

(p.getY() - q.getY()) \* Math.sqrt(3) / 6,

((p.getY() + q.getY()) / 2 +

(p.getX() - q.getX()) \* Math.sqrt(3) / 6));

Point2D t = new Point2D.Double(

(p.getX() + 2 \* q.getX()) / 3,

(p.getY() + 2 \* q.getY()) / 3);

Path2D path = new Path2D.Double();

path.moveTo(r.getX(), r.getY());

path.lineTo(s.getX(), s.getY());

path.lineTo(t.getX(), t.getY());

path.lineTo(r.getX(), r.getY());

path.closePath();

graph.setColor(color);

graph.fill(path);

drawKochCurve(p, r, n - 1, color);

drawKochCurve(r, s, n - 1, color);

drawKochCurve(s, t, n - 1, color);

drawKochCurve(t, q, n - 1, color);

}

private static void drawKochAntiSnowflake(Point2D c, double d,

int m, int n, Color color)

{

Point2D[] vs = new Point2D[m];

for (int i = 0; i < m; ++i)

{

vs[i] = new Point2D.Double(

c.getX() + d \* Math.cos(2 \* Math.PI / m \* i),

c.getY() + d \* Math.sin(2 \* Math.PI / m \* i));

}

Path2D path = new Path2D.Double();

path.moveTo(vs[0].getX(), vs[0].getY());

for (int i = 0; i < m; ++i)

{

path.lineTo(vs[(i + 1) % m].getX(), vs[(i + 1) % m].getY());

}

path.closePath();

graph.setColor(color);

graph.fill(path);

for (int i = 0; i < m; ++i)

{

drawKochCurve(vs[(i + 1) % m], vs[i], n, color);

}

}

public static void main(String[] args)

{

image = new BufferedImage(WIDTH, HEIGHT, BufferedImage.TYPE\_INT\_RGB);

graph = image.createGraphics();

graph.setColor(Color.WHITE);

graph.fill(new Rectangle2D.Double(0, 0, WIDTH, HEIGHT));

graph.setColor(Color.BLACK);

drawKochAntiSnowflake(new Point2D.Double(WIDTH / 2, HEIGHT / 2),

WIDTH / 3, 2, 4, Color.RED);

JFrame frame = new JFrame();

frame.addNotify();

frame.setSize(frame.getInsets().left +

frame.getInsets().right + WIDTH,

frame.getInsets().top +

frame.getInsets().bottom + HEIGHT);

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.add(new JPanel()

{

@Override

public void paintComponent(Graphics g)

{

Graphics2D G = (Graphics2D) g;

if (image != null)

{

G.drawImage(image, 0, 0, null);

}

}

});

frame.setVisible(true);

}

}

***Цветные мегаснежинки Коха***

package fractal;

import java.awt.Color;

import java.awt.Graphics;

import java.awt.Graphics2D;

import java.awt.geom.Line2D;

import java.awt.geom.Point2D;

import java.awt.geom.Rectangle2D;

import java.awt.image.BufferedImage;

import java.awt.geom.Path2D;

import javax.swing.JFrame;

import javax.swing.JPanel;

public class KochMegaSnowflakes

{

private static final int WIDTH = 640;

private static final int HEIGHT = 480;

private static BufferedImage image;

private static Graphics2D graph;

private static void drawKochCurve(Point2D p, Point2D q, int n, Color color)

{

if (n == 0)

{

graph.setColor(Color.BLACK);

graph.draw(new Line2D.Double(p, q));

return;

}

Point2D r = new Point2D.Double(

(2 \* p.getX() + q.getX()) / 3,

(2 \* p.getY() + q.getY()) / 3);

Point2D s = new Point2D.Double(

(p.getX() + q.getX()) / 2 -

(p.getY() - q.getY()) \* Math.sqrt(3) / 6,

((p.getY() + q.getY()) / 2 +

(p.getX() - q.getX()) \* Math.sqrt(3) / 6));

Point2D t = new Point2D.Double(

(p.getX() + 2 \* q.getX()) / 3,

(p.getY() + 2 \* q.getY()) / 3);

Path2D path = new Path2D.Double();

path.moveTo(r.getX(), r.getY());

path.lineTo(s.getX(), s.getY());

path.lineTo(t.getX(), t.getY());

path.lineTo(r.getX(), r.getY());

path.closePath();

graph.setColor(color);

graph.fill(path);

drawKochCurve(p, r, n - 1, color);

drawKochCurve(r, s, n - 1, color);

drawKochCurve(s, t, n - 1, color);

drawKochCurve(t, q, n - 1, color);

}

private static void drawKochSnowflake(Point2D c, double d,

int m, int n, Color color)

{

Point2D[] vs = new Point2D[m];

for (int i = 0; i < m; ++i) {

vs[i] = new Point2D.Double(

c.getX() + d \* Math.cos(2 \* Math.PI / m \* i),

c.getY() - d \* Math.sin(2 \* Math.PI / m \* i));

}

Path2D path = new Path2D.Double();

path.moveTo(vs[0].getX(), vs[0].getY());

for (int i = 0; i < m; ++i) {

path.lineTo(vs[(i + 1) % m].getX(), vs[(i + 1) % m].getY());

}

path.closePath();

graph.setColor(color);

graph.fill(path);

for (int i = 0; i < m; ++i) {

drawKochCurve(vs[(i + 1) % m], vs[i], n, color);

}

}

private static void drawKochMegaSnowflake(Point2D c, double d,

int m, int k, int n, Color bc, Color ec)

{

for (int i = 0; i < k; ++i)

{

Color color = new Color(

bc.getRed() + (ec.getRed() - bc.getRed()) \* i / k,

bc.getGreen() + (ec.getGreen() - bc.getGreen()) \* i / k,

bc.getBlue() + (ec.getBlue() - bc.getBlue()) \* i / k);

drawKochSnowflake(c, d \* (k - i) / k, m, n, color);

}

}

public static void main(String[] args)

{

image = new BufferedImage(WIDTH, HEIGHT, BufferedImage.TYPE\_INT\_RGB);

graph = image.createGraphics();

graph.setColor(Color.WHITE);

graph.fill(new Rectangle2D.Double(0, 0, WIDTH, HEIGHT));

graph.setColor(Color.BLACK);

drawKochMegaSnowflake(new Point2D.Double(WIDTH / 2, HEIGHT / 2),

WIDTH / 3, 5, 5, 5, Color.RED, Color.BLUE);

JFrame frame = new JFrame();

frame.addNotify();

frame.setSize(frame.getInsets().left +

frame.getInsets().right + WIDTH,

frame.getInsets().top +

frame.getInsets().bottom + HEIGHT);

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.add(new JPanel()

{

@Override

public void paintComponent(Graphics g)

{

Graphics2D G = (Graphics2D) g;

if (image != null)

{

G.drawImage(image, 0, 0, null);

}

}

});

frame.setVisible(true);

}

}

***Дерево Пифагора***

package fractal;

import java.awt.Color;

import java.awt.Graphics;

import java.awt.Graphics2D;

import java.awt.geom.Path2D;

import java.awt.geom.Rectangle2D;

import java.awt.image.BufferedImage;

import javax.swing.JFrame;

import javax.swing.JPanel;

public class PythagorasTree

{

private static final int WIDTH = 1000;

private static final int HEIGHT = 500;

private static BufferedImage image;

private static Graphics2D graph;

private static final double DA = Math.PI / 6;

private static final int BASE\_SIZE = 100;

private static final Color treeColor = new Color(0x712F26);

private static final Color leafColor = new Color(0x00FF00);

private static void drawPythagorasTree(double a, double size,

double x, double y)

{

double dx = size \* Math.sin(a);

double dy = size \* Math.cos(a);

double x1 = x;

double y1 = y;

double x2 = x + dx;

double y2 = y - dy;

double x3 = x + dx - dy;

double y3 = y - dy - dx;

double x4 = x - dy;

double y4 = y - dx;

double x5 = x - dy + size \* Math.cos(DA) \* Math.sin(a - DA);

double y5 = y - dx - size \* Math.cos(DA) \* Math.cos(a - DA);

Path2D path = new Path2D.Double();

path.moveTo(x1, y1);

path.lineTo(x2, y2);

path.lineTo(x3, y3);

path.lineTo(x5, y5);

path.lineTo(x4, y4);

path.lineTo(x1, y1);

path.closePath();

graph.setColor(getBetweenColor(leafColor, treeColor,

Math.pow(size / BASE\_SIZE, 0.2)));

graph.fill(path);

if (size > 1)

{

drawPythagorasTree(

a - DA,

size \* Math.cos(DA),

x4,

y4);

drawPythagorasTree(

a - DA + (Math.PI / 2),

size \* Math.sin(DA),

x5,

y5);

}

}

private static Color getBetweenColor(Color startColor, Color endColor,

double p)

{

return new Color(

(int) (startColor.getRed() +

(endColor.getRed() - startColor.getRed()) \* p),

(int) (startColor.getGreen() +

(endColor.getGreen() - startColor.getGreen()) \* p),

(int) (startColor.getBlue() +

(endColor.getBlue() - startColor.getBlue()) \* p));

}

public static void main(String[] args)

{

image = new BufferedImage(WIDTH, HEIGHT, BufferedImage.TYPE\_INT\_RGB);

graph = image.createGraphics();

graph.setColor(Color.WHITE);

graph.fill(new Rectangle2D.Double(0, 0, WIDTH, HEIGHT));

graph.setColor(Color.BLACK);

drawPythagorasTree(

Math.PI / 2,

BASE\_SIZE,

WIDTH / 2 - BASE\_SIZE / 2,

HEIGHT - 50);

JFrame frame = new JFrame();

frame.addNotify();

frame.setSize(frame.getInsets().left +

frame.getInsets().right + WIDTH,

frame.getInsets().top +

frame.getInsets().bottom + HEIGHT);

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.add(new JPanel()

{

@Override

public void paintComponent(Graphics g)

{

Graphics2D G = (Graphics2D) g;

if (image != null) {

G.drawImage(image, 0, 0, null);

}

}

});

frame.setVisible(true);

}

}