Оглавление

[**Введение** 2](#_Toc119848367)

[**1 Разработка приложения** 3](#_Toc119848368)

[**1.1 Разработка технического задания** 3](#_Toc119848369)

[*1.1.1 Полное наименование системы и её условное обозначение* 3](#_Toc119848370)

[*1.1.2 Основание для разработки* 3](#_Toc119848371)

[*1.1.3 Назначение и цели создания системы* 3](#_Toc119848372)

[**1.2 Анализ технического задания** 4](#_Toc119848373)

[*1.2.1 Общие положения* 4](#_Toc119848374)

[*1.2.2 Требования к системе* 4](#_Toc119848375)

[**1.3 Стандарты** 4](#_Toc119848376)

[**1.4 Требования к техническим средствам** 5](#_Toc119848377)

[**2 Разработка модели предметной области** 6](#_Toc119848378)

[**2.1 Анализ предметной области** 6](#_Toc119848379)

[**2.2 Разработка структуры классов** 6](#_Toc119848380)

[**3 Разработка структуры приложения** 7](#_Toc119848381)

[**3.1 Разработка архитектуры** 7](#_Toc119848382)

[**3.2 Проектирование пользовательского интерфейса и взаимодействие с ним** 7](#_Toc119848383)

[**Заключение** 11](#_Toc119848384)

[**Список литературы** 12](#_Toc119848385)

[**Приложения** 13](#_Toc119848386)

[**Приложение 1: Код класса *FractalsFrame*** 13](#_Toc119848387)

[**Приложение 2: Код класса *MyLine***. 13](#_Toc119848388)

[**Приложение 3: Код класса *CantorFractal***. 13](#_Toc119848389)

[**Приложение 4: Код класса *LevyFractalPanel*** 13](#_Toc119848390)

[**Приложение 5: Код класса Minkowski*FractalPanel***. 13](#_Toc119848391)

[**Приложение 6: Код класса *SerpinskiyFractalPanel*** 13](#_Toc119848392)

[**Приложение 7: Код класса *Main*** 13](#_Toc119848393)

**Введение**

В данной пояснительной записке рассматривается описание приложения «Планировщик задач» на основе объектно-ориентированного подхода.

При объектно-ориентированном подходе программа представляет собой описание объектов, их свойств (или атрибутов), совокупностей (или классов), отношений между ними, способов их взаимодействия и операций над объектами (или методов).

Несомненным преимуществом данного подхода является концептуальная близость к предметной области произвольной структуры и назначения. Механизм наследования атрибутов и методов позволяет строить производные понятия на основе базовых и таким образом создавать модель сколь угодно сложной предметной области с заданными свойствами.

Еще одним теоретически интересным и практически важным свойством объектно-ориентированного подхода является поддержка механизма обработки событий, которые изменяют атрибуты объектов и моделируют их взаимодействие в предметной области.

В отличие от других подходов к программированию, объектно-ориентированный подход требует глубокого понимания основных принципов, или концепций, на которых он базируется.

К числу основополагающих понятий ООП обычно относят абстракцию данных, наследование, инкапсуляцию и полиморфизм.

Объектно-ориентированное программирование в настоящее время является абсолютным лидером в области прикладного программирования.

В качестве основного инструмента разработки применяется IntelliJ IDEA v 2022.2.3. Язык программирования Java.

**1 Разработка приложения**

**1.1 Разработка технического задания**

*1.1.1 Полное наименование системы и её условное обозначение*

Полное наименование: Разработка приложения «Отрисовщик фракталов» для последовательного рисунка части геометрических фигур с характерными для них свойствами.

Условное обозначение системы: «Отрисовщик фракталов».

*1.1.2 Основание для разработки*

Основанием для разработки данной информационной системы является приказ по НГТУ на курсовое проектирование по дисциплине «Программирование на языке Java».

*1.1.3 Назначение и цели создания системы*

1.1.3.1 Назначение системы

Программный продукт предназначен для пошагового построения фрактала на экране с целью изучения его свойств.

1.1.3.2 Цели создания системы

Программный продукт разрабатывается с целью:

1) изучения свойств геометрических фигур;

2) изучения построения сложных объектов из простых элементов;

3) поиска новых объектов при помощи математических операций;

4) получения нужных фракталов в одном приложении.

С помощью приложения пользователь сможет достигнуть вышеперечисленные цели.

1.1.3.3 Основные задачи разработки

1) Обеспечить выбор пошагового отображения фрактала пользователем;

2) Обеспечить выбор по отображению нужного фрактала пользователем;

3) Обеспечить сброс в начальное положение отображения фрактала при клике на кнопку.

**1.2 Анализ технического задания**

*1.2.1 Общие положения*

Согласно техническому заданию необходимо разработать приложение для компьютера, которое предназначено для пошагового построения геометрических фигур.

*1.2.2 Требования к системе*

1.2.2.1 Требования к структуре и функционированию системы

Программный продукт, разрабатываемый в рамках курсового проекта, должен удовлетворять следующему перечню функциональных требований:

1. Удобный и понятный графический интерфейс
2. Сброс выбранного фрактала в начальное положение
3. Возможность выбрать фрактал для его отображения

1.2.2.2 Входные данные

Входными данными при работе с программным продуктом должны быть нажатия на кнопку мыши (левая) и нажатия на клавиатуру.

1.2.2.3 Выходные данные

Выходными данными при работе программы являются данные, выводимые на экран пользователя.

1.2.2.4 Требования к надёжности

Система должна сохранять работоспособность и обеспечивать восстановление своих функций при возникновении внештатных ситуаций.

1.2.2.5 Требования к эргономике и технической эстетике

Подсистемы ввода данных, а также формирования и визуализации отчетности должны обеспечивать удобный для конечного пользователя интерфейс.

Главное окно программного продукта должно позволять пользователю выбрать нужный фрактал для построения.

**1.3 Стандарты**

Программный продукт разрабатывается на основании следующих государственных стандартов:

1. 2.103-68 ЕСКД. Стадии разработок
2. 2.104-68 ЕСКД. Основные надписи
3. 2.105-95 ЕСКД. Общие требования к текстовым документам
   1. -96 ЕСКД. Текстовые документы
4. 2.111- 68 ЕСКД. Общие требования к текстовым документам.
5. 2.118-73 ЕСКД. Техническое предложение
6. 2.120-73 ЕСКД. Технический проект
7. 2.316-68 ЕСКД. Правила нанесения на чертежах надписей, технических требований и таблиц
8. 7.1-2003 Система стандартов по информации, библиотечному и издательскому делу. Библиографическая запись. Библиографическое описание. Общие требования и правила составления.

**1.4 Требования к техническим средствам**

Для удобства работы система должна обеспечивать отображение *GUI* с расширенным набором пользовательских элементов, что соответствует платформе *Java*, IntelliJ IDEA v 2022.2.3 не ниже версии 11.0.2.

Минимальные технические характеристики компьютера, на котором гарантируется стабильная работа программы:

1. компьютер/процессор: компьютер-терминал с процессором класса *Intel* *Core i7* с тактовой частотой 2,3 ГГц и новее;
2. память: не менее 512 МБ ОЗУ;
3. монитор: монитор с разрешением 1200x600 точек или более высоким;
4. наличие свободного дискового пространства на жёстком диске не менее 20 Гб.

**2 Разработка модели предметной области**

**2.1 Анализ предметной области**

Программируемое приложение должно позволять пользователю выбирать и пошагово отображать выбранный фрактал, а также возможность возвращать фрактал в начальное положение при клике на кнопку «Сброс».

Нефункциональные требования к программному средству:

1. эффективность: программа должна иметь минимальные требования к аппаратному обеспечению. Не должно требоваться дополнительных периферийных средств (сканер, дополнительные дисководы и т.д.);
2. понятность: программа должна быть интуитивно понятна, иметь удобный пользовательский интерфейс.

**2.2 Разработка структуры классов**

В соответствие с анализом предметной области можно выделить основные классы информационной системы и их атрибуты:

1. Главный класс *FractalsFrame* содержит основу рабочего окна:

* Методы отрисовки компонентов на форме;
* Методы обработки взаимодействия пользователя с интерфейсом;

1. Класс *MyLine* – содержит необходимые переменные-координаты для отрисовки фракталов;
2. Класс *CantorFractal* – содержит необходимые алгоритмы для отрисовки фрактала Кантора;
3. Класс *LevyFractalPanel* – содержит необходимые алгоритмы для отрисовки фрактала Леви;
4. Класс Minkowski*FractalPanel* – содержит необходимые алгоритмы для отрисовки фрактала Миньковского;
5. Класс *SerpinskiyFractalPanel* – содержит необходимые алгоритмы для отрисовки фрактала треугольника Серпинского;

**3 Разработка структуры приложения**

**3.1 Разработка архитектуры**

Для Программы «Отрисовщик фракталов» необходимо несколько классов с описаниями различных алгоритмов построения и отображения геометрических фигур и несколько классов для вывода полученной фигуры на экран.

Изначально перед пользователем открывается класс *FractalsFrame*, содержащий панель со списком названий фракталов, которые предложены пользователю для последовательного построения. По нажатию на один из них, на экране появляется начальное положение геометрической фигуры. Пользователю предоставляется выбор по началу построения фрактала автоматически с интервалом между отображением новой геометрической фигуры в 1 секунду, выполнению следующей операции построения фрактала по алгоритму и сброс, который возвращает фрактал в начальное положение. В любой момент пользователь может выбрать последовательное отображение фрактала из списка. При переходе к построению другого фрактала предыдущий сохраняет свою позицию.

**3.2 Проектирование пользовательского интерфейса и взаимодействие с ним**

В данном проекте для продуктивной работы реализован удобный пользовательский интерфейс (рис. 1, рис. 2).

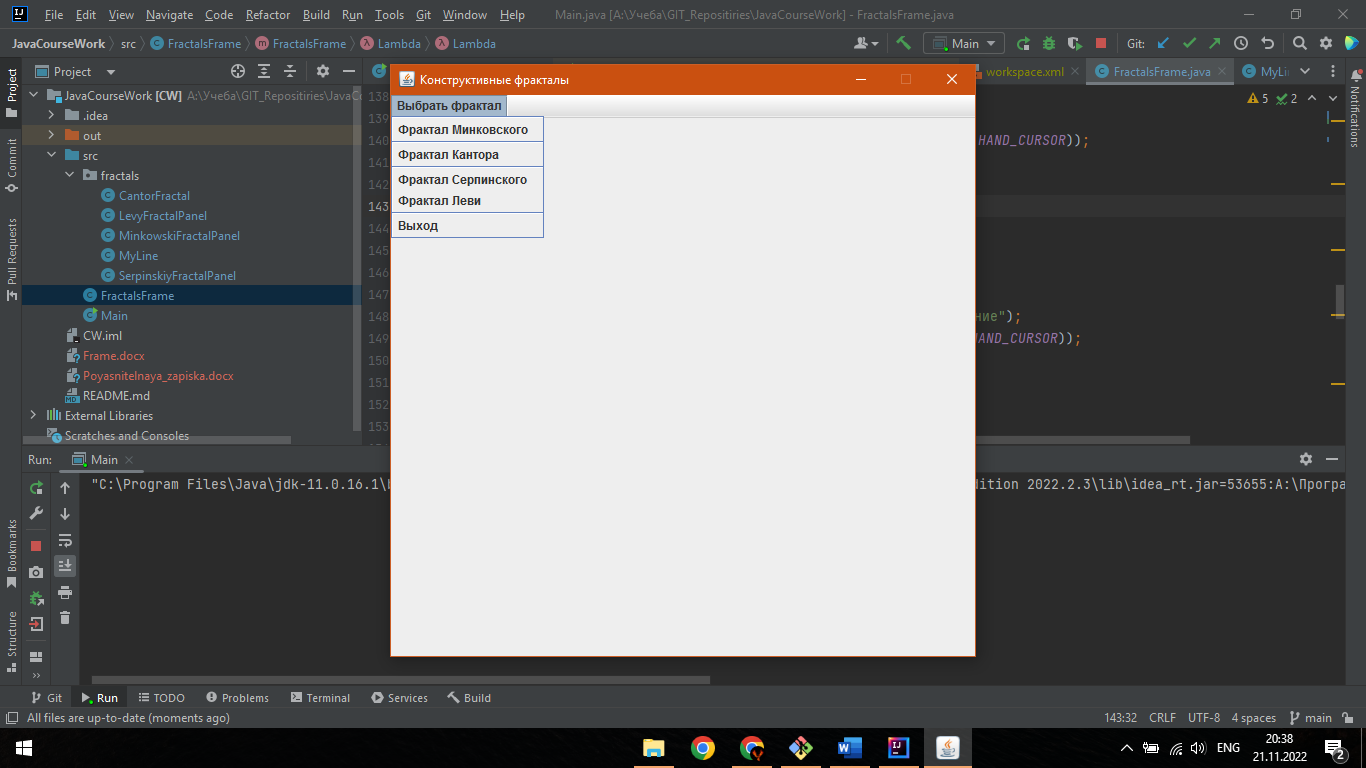


Рис. 1 – Интерфейс приложения.

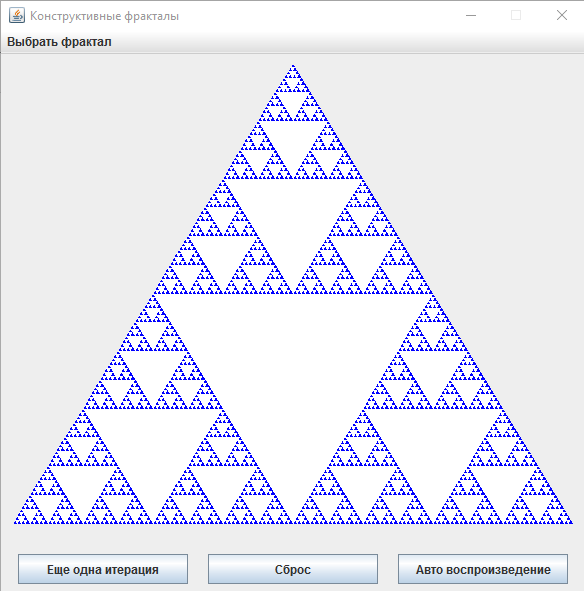


Рис.2 – Интерфейс приложения

Основной сценарий работы пользователя с приложением:

1. Пользователь видит кнопку «Выбрать фрактал» и нажимает на нее для последовательного построения фрактала. Появляется выпадающий список с названиями фракталов, который пользователь может построить.
2. При необходимости привести геометрическую фигуру в начальное положение пользователь нажимает кнопку «Сброс».
3. При необходимости автоматического построения фрактала пользователь нажимает на кнопку «Автовоспроизведение»
4. При желании самостоятельно построить следующее отображение фрактала пользователь нажимает на кнопку «Еще одна итерация»
5. При необходимости построить другой фрактал пользователь выбирает его из выпадающего списка в верхней панели.

**Заключение**

В данной курсовой работе было разработано приложение «Отрисовщик фракталов» на основе объектно-ориентированного подхода. Этот метод позволяет обеспечить разделение данных, управление доступом к ним, а также расширяемость и модифицируемость.

В проекте удалось реализовать все поставленные задачи:

1. Создать удобный и понятный графический интерфейс;
2. Создать последовательное отображение фрактала;
3. Реализовать возможность выполнять построение геометрических фигур автоматически;
4. Реализовать возвращение фрактала в начальное положение;
5. Реализовать выбор по построению нужного фрактала;

Таким образом, было реализовано приложение «Отрисовщик фракталов», которое позволяет пользователю строить сложные геометрические фигуры из простейших при помощи определенных алгоритмов и изучать их свойства на каждом этапе построения.

**Список литературы**

1. <https://ru.wikipedia.org/wiki/%D0%A4%D1%80%D0%B0%D0%BA%D1%82%D0%B0%D0%BB>
2. <https://piziadas.com/ru/>
3. <https://habr.com/ru/>
4. <https://stackoverflow.com/>
5. https://github.com/

**Приложения**

**Приложение 1: Код класса *FractalsFrame*.**

import fractals.CantorFractal;

import fractals.LevyFractalPanel;

import fractals.MinkowskiFractalPanel;

import fractals.SerpinskiyFractalPanel;

import javax.swing.\*;

import java.awt.\*;

import java.util.concurrent.atomic.AtomicBoolean;

public class FractalsFrame extends JFrame {

private final static int sizeOfFrame = 600;

private final AtomicBoolean key = new AtomicBoolean(false);

FractalsFrame() {

JMenu fileMenu = new JMenu("Выбрать фрактал");

//Фрактал Леви

JMenuItem LevyFractalItem = new JMenuItem("Фрактал Леви");

LevyFractalItem.addActionListener(e -> {

LevyFractalPanel panel = new LevyFractalPanel();

panel.setLayout(null);

// кнопка update

JButton updateButton = new JButton("Еще одна итерация");

updateButton.setCursor(Cursor.getPredefinedCursor(Cursor.HAND\_CURSOR));

updateButton.setFocusable(false);

updateButton.addActionListener(e13 -> {

panel.flagToContinue = true; //позволяем рисовать

panel.stepIncrement(); // увеличиваем шаг

panel.repaint(); // рисуем

});

// кнопка reset

JButton resetButton = new JButton("Сброс");

resetButton.setCursor(Cursor.getPredefinedCursor(Cursor.HAND\_CURSOR));

resetButton.setFocusable(false);

resetButton.addActionListener(e1 -> {

key.set(false); // останавливаем авто воспроизведение если запущено

panel.setNullStep(); // обнуляем панель

panel.flagToContinue = true; // позволяем рисовать

panel.repaint(); // рисуем с ноля

});

// кнопка auto

JButton autoButton = new JButton("Авто воспроизведение");

autoButton.setCursor(Cursor.getPredefinedCursor(Cursor.HAND\_CURSOR));

autoButton.setFocusable(false);

autoButton.addActionListener(e1 -> new Thread(() -> {

key.set(true); // false установит кнопка reset, а пока авто воспроизведение работает будет true

while (key.get()) {

try {

Thread.sleep(1000); // спим 1 сек

if (!key.get()) return; // если нажали reset, то выходим

panel.stepIncrement(); // увеличиваем шаг

panel.flagToContinue = true; // позволяем рисовать

panel.repaint(); // рисуем

if (panel.getStep() > 20) panel.setNullStep(); // если более 20 шагов, обнуляем рисунок и начинаем заново

} catch (InterruptedException ignored) {

return;

}

}

}).start());

panel.add(updateButton).setBounds(17, 500, 170, 30);

panel.add(resetButton).setBounds(207, 500, 170, 30);

panel.add(autoButton).setBounds(397, 500, 170, 30);

getContentPane().removeAll();

getContentPane().add(panel);

pack();

});

JMenuItem minkowskiFractalItem = new JMenuItem("Фрактал Минковского");

minkowskiFractalItem.addActionListener(e -> {

MinkowskiFractalPanel panel = new MinkowskiFractalPanel();

panel.setLayout(null);

JButton updateButton = new JButton("Еще одна итерация");

updateButton.setCursor(Cursor.getPredefinedCursor(Cursor.HAND\_CURSOR));

updateButton.setFocusable(false);

updateButton.addActionListener(e13 -> {

panel.flagToContinue = true;

panel.stepIncrement();

panel.repaint();

});

JButton resetButton = new JButton("Сброс");

resetButton.setCursor(Cursor.getPredefinedCursor(Cursor.HAND\_CURSOR));

resetButton.setFocusable(false);

resetButton.addActionListener(e1 -> {

key.set(false);

panel.setNullStep();

panel.flagToContinue = true;

panel.repaint();

});

JButton autoButton = new JButton("Авто воспроизведение");

autoButton.setCursor(Cursor.getPredefinedCursor(Cursor.HAND\_CURSOR));

autoButton.setFocusable(false);

autoButton.addActionListener(e1 -> new Thread(() -> {

key.set(true);

while (key.get()) {

try {

Thread.sleep(1000);

if (!key.get()) return;

panel.stepIncrement();

panel.flagToContinue = true;

panel.repaint();

if (panel.getStep() > 6) panel.setNullStep();

} catch (InterruptedException ignored) {

return;

}

}

}).start());

panel.add(updateButton).setBounds(17, 500, 170, 30);

panel.add(resetButton).setBounds(207, 500, 170, 30);

panel.add(autoButton).setBounds(397, 500, 170, 30);

getContentPane().removeAll();

getContentPane().add(panel);

pack();

});

JMenuItem cantorFractalItem = new JMenuItem("Фрактал Кантора");

cantorFractalItem.addActionListener(e -> {

CantorFractal cantorPanel = new CantorFractal();

cantorPanel.setLayout(null);

JButton updateButton = new JButton("Еще одна итерация");

updateButton.setCursor(Cursor.getPredefinedCursor(Cursor.HAND\_CURSOR));

updateButton.setFocusable(false);

updateButton.addActionListener(e1 -> {

cantorPanel.stepIncrement();

cantorPanel.repaint();

});

JButton resetButton = new JButton("Сброс");

resetButton.setCursor(Cursor.getPredefinedCursor(Cursor.HAND\_CURSOR));

resetButton.setFocusable(false);

resetButton.addActionListener(e1 -> {

key.set(false);

cantorPanel.setNullStep();

cantorPanel.repaint();

});

JButton autoButton = new JButton("Авто воспроизведение");

autoButton.setCursor(Cursor.getPredefinedCursor(Cursor.HAND\_CURSOR));

autoButton.setFocusable(false);

autoButton.addActionListener(e1 -> new Thread(() -> {

key.set(true);

while (key.get()) {

try {

Thread.sleep(1000);

if (!key.get()) return;

cantorPanel.stepIncrement();

System.out.print("Before repaint = " + cantorPanel.getStep() + " ");

cantorPanel.repaint();

if (cantorPanel.getStep() == 14) {

cantorPanel.setNullStep();

}

} catch (InterruptedException ignored) {

return;

}

}

}).start());

cantorPanel.add(updateButton).setBounds(17, 500, 170, 30);

cantorPanel.add(resetButton).setBounds(207, 500, 170, 30);

cantorPanel.add(autoButton).setBounds(397, 500, 170, 30);

getContentPane().removeAll();

getContentPane().add(cantorPanel);

pack();

});

JMenuItem serpinskiyTriangleItem = new JMenuItem("Фрактал Серпинского");

serpinskiyTriangleItem.addActionListener(e -> {

SerpinskiyFractalPanel serpinskiyPanel = new SerpinskiyFractalPanel();

serpinskiyPanel.setLayout(null);

getContentPane().removeAll();

getContentPane().add(serpinskiyPanel);

JButton updateButton = new JButton("Еще одна итерация");

updateButton.setCursor(Cursor.getPredefinedCursor(Cursor.HAND\_CURSOR));

updateButton.setFocusable(false);

updateButton.addActionListener(e12 -> {

serpinskiyPanel.stepIncrement();

serpinskiyPanel.repaint();

});

JButton resetButton = new JButton("Сброс");

resetButton.setCursor(Cursor.getPredefinedCursor(Cursor.HAND\_CURSOR));

resetButton.setFocusable(false);

resetButton.addActionListener(e1 -> {

key.set(false);

serpinskiyPanel.setNullStep();

serpinskiyPanel.repaint();

});

JButton autoButton = new JButton("Авто воспроизведение");

autoButton.setCursor(Cursor.getPredefinedCursor(Cursor.HAND\_CURSOR));

autoButton.setFocusable(false);

autoButton.addActionListener(e1 -> new Thread(() -> {

key.set(true);

while (key.get()) {

try {

Thread.sleep(1000);

if (!key.get()) return;

} catch (InterruptedException interruptedException) {

return;

}

serpinskiyPanel.stepIncrement();

serpinskiyPanel.repaint();

if (serpinskiyPanel.getStep() == 10) serpinskiyPanel.setNullStep();

}

}).start());

serpinskiyPanel.add(updateButton).setBounds(17, 500, 170, 30);

serpinskiyPanel.add(resetButton).setBounds(207, 500, 170, 30);

serpinskiyPanel.add(autoButton).setBounds(397, 500, 170, 30);

pack();

});

JMenuItem exitItem = new JMenuItem("Выход");

exitItem.addActionListener(e -> System.exit(0));

fileMenu.add(minkowskiFractalItem);

fileMenu.addSeparator();

fileMenu.add(cantorFractalItem);

fileMenu.addSeparator();

fileMenu.add(serpinskiyTriangleItem);

fileMenu.add(LevyFractalItem);

fileMenu.addSeparator();

fileMenu.add(exitItem);

JMenuBar menuBar = new JMenuBar();

menuBar.add(fileMenu);

setJMenuBar(menuBar);

setPreferredSize(new Dimension(sizeOfFrame, sizeOfFrame)); // задаем размеры окна

pack();

}

}

**Приложение 2: Код класса *MyLine*.**

package fractals;

public class MyLine {

public int x;

public int y;

public int X;

public int Y;

public MyLine(int x, int y, int X, int Y) {

this.x = x;

this.y = y;

this.X = X;

this.Y = Y;

}

}

**Приложение 3: Код класса *CantorFractal*.**

package fractals;

import javax.swing.\*;

import java.awt.\*;

public class CantorFractal extends JPanel {

public final int paddingX = 170;

public final int paddingY = 15;

public final int height = 25;

public int step = 0;

public final int width = 243;

@Override

public void paintComponent(Graphics g) {// метод рисования Фрактала

super.paintComponent(g);

g.setColor(Color.BLACK); // устанавливаем цвет рисования

paintItems(g, width, paddingX, paddingY, step); // вызываем рекурсивный метод отрисовки шагов

g.fillRect(paddingX, paddingY, width, height); // отрисовка первого шага

}

private void paintItems(Graphics g, int width, int paddingX, int paddingY, int step) {// рекурсивный метод отрисовки шагов

if (this.step > 13) this.step--; // Более 13 шагов рисовать не нужно, убираем на единицу назад

if (step == 0) return;

g.fillRect(paddingX, paddingY + height + 7, width > 1 ? width / 3 : 1, height);// рисуем одну сторону

g.fillRect(paddingX + (width / 3) \* 2, paddingY + height + 7, width > 1 ? width / 3 : 1, height); //рисуем вторую сторону

step--;

paintItems(g, width / 3, paddingX, paddingY + height + 7, step); //рекурсивный вызов

paintItems(g, width / 3, paddingX + (width / 3) \* 2, paddingY + height + 7, step); //рекурсивный вызов

}

public void stepIncrement() {

step++;

} // метод для увеличения шагов

public void setNullStep() {

step = 0;

} // метод обнуления шагов для кнопки "авто воспроизведение'

public int getStep() {

return step;

} // getter

}

**Приложение 4: Код класса *LevyFractalPanel*.**

package fractals;

import javax.swing.\*;

import java.awt.\*;

import java.util.ArrayList;

public class LevyFractalPanel extends JPanel {

public static int sizeOfPanel = 590;

private int step = 0;

public boolean flagToContinue = true;

public static ArrayList<MyLine> lines = new ArrayList<>();

public static double[][] pattern = {

{0, 0},

{0.5, 0.5},

{1, 0}

};

static Point A = new Point(2 \* sizeOfPanel / 3, sizeOfPanel / 2);

static Point B = new Point(sizeOfPanel / 3, sizeOfPanel / 2);

public void draw() {

if (flagToContinue) {

if (lines.size() == 0) {

lines.add(new MyLine(A.x, A.y, B.x, B.y));

//выключаем необходимость продолжения

flagToContinue = false;

return;

}

ArrayList<MyLine> bufferLines = new ArrayList<>(); // буферный лист, сюда мы будем записывать новые линии

ArrayList<Point> bufferPoints = new ArrayList<>(); // буферный лист точек, которые мы получаем

// перебираем все линии и преобразовываем их

for (MyLine line :

lines) {

bufferPoints.clear(); // очищаем лист точек, так как иначе он заполнится не нужными точками от предыдущих линий

for (double[] doubles : pattern) {

double xRes = (line.X - line.x) \* doubles[0] - (line.Y - line.y) \* doubles[1] + line.x;

double yRes = (line.Y - line.y) \* doubles[0] + (line.X - line.x) \* doubles[1] + line.y;

bufferPoints.add(new Point((int) xRes, (int) yRes)); // получи точку запоминаем ее

}

//в этом цикле проходим по существующим точкам и создаем линии, добавляем их в буфер линий

for (int i = 0; i < bufferPoints.size() - 1; i++) {

bufferLines.add(new MyLine(bufferPoints.get(i).x,

bufferPoints.get(i).y,

bufferPoints.get(i + 1).x,

bufferPoints.get(i + 1).y));

}

}

flagToContinue = false; // отключаем флаг прохода

if (step > 20) return;

lines = bufferLines; // забываем про старые линии, так как они не актуальны, и запоминаем новые

}

}

@Override

protected void paintComponent(Graphics g) { // этот метод вызывается у компонента при каждом обновлении кадра

super.paintComponent(g);

this.setBackground(Color.WHITE);

draw();

//цикл перебирает все существующие линии и рисует их

for (MyLine line : lines) {

g.drawLine(line.x, line.y, line.X, line.Y);

}

}

public void stepIncrement() {

step++;

}

public void setNullStep() {

step = 0;

lines.clear();

}

public int getStep() {

return step;

}

}

**Приложение 5: Код класса Minkowski*FractalPanel*.**

package fractals;

import java.awt.\*;

import java.util.ArrayList;

import javax.swing.\*;

public class MinkowskiFractalPanel extends JPanel {

public static final int sizeOfPanel = 590;

private int step = 0;

public boolean flagToContinue = true;

public static ArrayList<MyLine> lines = new ArrayList<>();

public static double[][] pattern = {

{0, 0},

{0.25, 0},

{0.25, -0.25},

{0.5, -0.25},

{0.5, 0.25},

{0.75, 0.25},

{0.75, 0},

{1, 0}

};

static Point A = new Point(150, 105);

static int size = sizeOfPanel / 2;

public void draw() {

if (flagToContinue) {

if (lines.size() == 0) {

lines.add(new MyLine(A.x, A.y, A.x + size, A.y));

lines.add(new MyLine(A.x + size, A.y, A.x + size, A.y + size));

lines.add(new MyLine(A.x + size, A.y + size, A.x, A.y + size));

lines.add(new MyLine(A.x, A.y + size, A.x, A.y));

//выключаем необходимость продолжения

flagToContinue = false;

return;

}

ArrayList<MyLine> bufferLines = new ArrayList<>(); // буферный лист, сюда мы будем записывать новые линии

ArrayList<Point> bufferPoints = new ArrayList<>(); // буферный лист точек, которые мы получаем

// перебираем все линии и преобразовываем их

for (MyLine line :

lines) {

bufferPoints.clear(); // очищаем лист точек, так как иначе он заполнится не нужными точками от предыдущих линий

for (double[] doubles : pattern) {

double xRes = (line.X - line.x) \* doubles[0] - (line.Y - line.y) \* doubles[1] + line.x;

double yRes = (line.Y - line.y) \* doubles[0] + (line.X - line.x) \* doubles[1] + line.y;

bufferPoints.add(new Point((int) xRes, (int) yRes)); // получи точку запоминаем ее

}

//в этом цикле проходим по существующим точкам и создаем линии, добавляем их в буфер линий

for (int i = 0; i < bufferPoints.size() - 1; i++) {

bufferLines.add(new MyLine(bufferPoints.get(i).x,

bufferPoints.get(i).y,

bufferPoints.get(i + 1).x,

bufferPoints.get(i + 1).y));

}

}

flagToContinue = false; // отключаем флаг прохода

if (step > 6) return;

lines = bufferLines; // забываем про старые линии, так как они не актуальны, и запоминаем новые

}

}

@Override

protected void paintComponent(Graphics g) { // этот метод вызывается у компонента при каждом обновлении кадра

super.paintComponent(g);

this.setBackground(Color.WHITE);

draw();

//цикл перебирает все существующие линии и рисует их

for (MyLine line : lines) {

g.drawLine(line.x, line.y, line.X, line.Y);

}

}

public void stepIncrement() {

step++;

}

public void setNullStep() {

step = 0;

lines.clear();

}

public int getStep() {

return step;

}

}

**Приложение 6: Код класса *SerpinskiyFractalPanel*.**

package fractals;

import javax.swing.\*;

import java.awt.\*;

public class SerpinskiyFractalPanel extends JPanel {

private int step = 0;

@Override

protected void paintComponent(Graphics g) {// метод рисования Фрактала

super.paintComponent(g);

if (step == 0) {

g.setColor(Color.BLUE); // устанавливаем цвет рисования

g.fillPolygon(new int[]{12, 292, 572}, new int[]{470, 10, 470}, 3);// рисуем первый треугольник

}

draw(step, new int[]{12, 292, 572}, new int[]{470, 10, 470}, g); // вызываем рекурсивный метод отрисовки шагов

}

private void draw(int n, int[] x, int[] y, Graphics gc) {// рекурсивный метод отрисовки шагов

if (this.step > 9) this.step--; // Более 13 шагов рисовать не нужно, убираем на единицу назад

if (n > 0) {

// высчитываем точки

int x1 = (x[0] + x[1]) / 2;

int y1 = (y[0] + y[1]) / 2;

int x2 = (x[1] + x[2]) / 2;

int y2 = (y[1] + y[2]) / 2;

int x3 = (x[2] + x[0]) / 2;

int y3 = (y[2] + y[0]) / 2;

// отрисовка первого треугольника

gc.setColor(Color.BLUE);

gc.fillPolygon(x, y, 3);

// отрисовка следующего шага

gc.setColor(Color.WHITE);

gc.fillPolygon(new int[]{x1, x2, x3}, new int[]{y1, y2, y3}, 3);

// рекурсивный вызов

draw(n - 1, new int[]{x[0], x1, x3}, new int[]{y[0], y1, y3}, gc);

draw(n - 1, new int[]{x1, x[1], x2}, new int[]{y1, y[1], y2}, gc);

draw(n - 1, new int[]{x3, x2, x[2]}, new int[]{y3, y2, y[2]}, gc);

}

}

public int getStep() {

return step;

}

public void stepIncrement() {

step++;

}// метод для увеличения шагов

public void setNullStep() {

step = 0;

}// метод обнуления шагов для кнопки "авто воспроизведение'

}

**Приложение 7: Код класса *Main*.**

import javax.swing.\*;

import java.awt.\*;

public class Main {

public static void main(String[] args) {

EventQueue.invokeLater(() ->{

JFrame frame = new FractalsFrame();

frame.setTitle("Конструктивные фракталы");

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

frame.setLocationRelativeTo(null);

frame.setVisible(true);

frame.setResizable(false);

});

}

}