Оглавление

[**Введение** 2](#_Toc123214576)

[**1 Разработка приложения** 3](#_Toc123214577)

[**1.1 Разработка технического задания** 3](#_Toc123214578)

[*1.1.1 Полное наименование системы и её условное обозначение* 3](#_Toc123214579)

[*1.1.2 Основание для разработки* 3](#_Toc123214580)

[*1.1.3 Назначение и цели создания системы* 3](#_Toc123214581)

[**1.2 Анализ технического задания** 4](#_Toc123214582)

[*1.2.1 Общие положения* 4](#_Toc123214583)

[*1.2.2 Требования к системе* 4](#_Toc123214584)

[**1.3 Стандарты** 5](#_Toc123214585)

[**1.4 Требования к техническим средствам** 5](#_Toc123214586)

[**2 Разработка модели предметной области** 7](#_Toc123214587)

[**2.1 Анализ предметной области** 7](#_Toc123214588)

[**2.2 Разработка структуры классов** 7](#_Toc123214589)

[**3 Разработка структуры приложения** 8](#_Toc123214590)

[**3.1 Разработка архитектуры** 8](#_Toc123214591)

[**3.2 Проектирование пользовательского интерфейса и взаимодействие с ним** 8](#_Toc123214592)

[**Заключение** 12](#_Toc123214593)

[**Список литературы** 13](#_Toc123214594)

[**Приложения** 14](#_Toc123214595)

[**Приложение 1: Код класса *FractalsFrame*.** 14](#_Toc123214596)

[**Приложение 2: Код класса *MyLine*.** 14](#_Toc123214597)

[**Приложение 3: Код класса *CantorFractal*.** 14](#_Toc123214598)

[**Приложение 4: Код класса *LevyFractalPanel*.** 14](#_Toc123214599)

[**Приложение 5: Код класса Minkowski*FractalPanel*.** 14](#_Toc123214600)

[**Приложение 6: Код класса *SerpinskiyFractalPanel*.** 14](#_Toc123214601)

[**Приложение 7: Код класса *Main*.** 14](#_Toc123214602)

**Введение**

В данной пояснительной записке рассматривается описание приложения отрисовщик фракталов на основе объектно-ориентированного подхода с применением графического интерфейса Java Swing.

Swing в Java – это легкий инструментарий с графическим интерфейсом, который имеет широкий спектр виджетов для создания оптимизированных оконных приложений. Это часть JFC (Java Foundation Classes). Он построен на основе AWT API и полностью написан на Java. Он не зависит от платформы в отличие от AWT и имеет легкие компоненты. Swing не зависит от платформы, модель – представление – контроллер "GUI фреймворк для Java, который следует однопоточной модели программирования. Кроме того, эта структура обеспечивает уровень абстракции между структурой кода и графическим представлением графического интерфейса на основе Swing.

Объектно-Ориентированный подход позволяет создавать приложение, которое в любой момент может быть расширено по функционалу. Кроме того, правильная архитектура приложения помогает разработчику ориентироваться в коде, использовать зависимости и не допускать ошибок. Механизм наследования позволяет строить производные понятия на основе базовых и таким образом создавать модель сколь угодно сложной предметной области с заданными свойствами.

Существует четыре основных принципа объектно-ориентированного программирования: абстракция, инкапсуляция, наследование, полиморфизм.

Благодаря разработке приложения на языке программирования Java, разработчик получает долгосрочную поддержку и развитие проекта, так как Java поддерживает обратную совместимость с более ранними версиями и имеет огромную базу решений задач и проблем в интернете.

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.

5) поиска новых математических алгоритмов, которые «рисуют» фигуры.

6) наглядного просмотра работы математических алгоритмов.

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.

4) Обеспечить отображение фрактала в нужной позиции.

5) Обеспечить диапазон шагов рисунка фрактала.

6) Обеспечить выбор фрактала в окне.

**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, обладающими множеством самоподобия.

*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
4. Возможность отобразить нужную позицию фрактала

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, содержать поле для ввода диапазона позиций фрактала и поле ввода для отображения нужной позиции геометрической фигуры.

**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 i3 первого поколения, Intel Pentium* или выше.
2. память: не менее 512 МБ ОЗУ;
3. монитор: монитор с разрешением 800x600 точек или более высоким;
4. Операционная система Windows XP или выше.

**2 Разработка модели предметной области**

**2.1 Анализ предметной области**

Наиболее важными при определении целей проекта создания приложения для последовательного рисунка геометрических фигур является получение полное понимание математических алгоритмов, при помощи которых строятся фракталы. Необходимо проанализировать все особенности правил построения фигур и используемые программно-аппаратные средства. Только после полного анализа всех элементов, возможно успешное определение целей проекта и эффективное планирование дальнейших действий.

Большинство аналитиков, ученых, учителей и учеников нуждаются в приложении, которое обладает низкими техническими требованиями и отображает последовательное отображение фракталов с возможностью выбора нужной позиции. Приложение позволяет пользователям выбирать нужный фрактал и наглядно изучать свойства геометрических фигур. Удобное приложение, созданное для сотрудников, облегчит задачу и позволит экономить время.

**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).

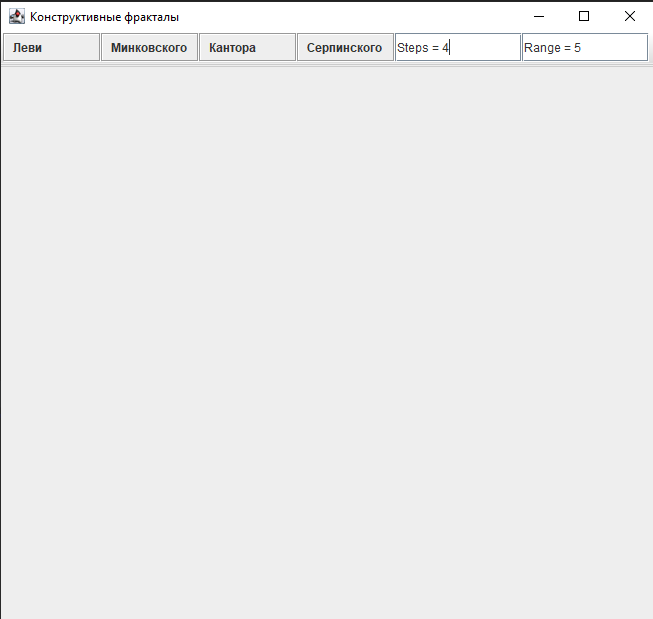


Рисунок 1 – Интерфейс приложения.

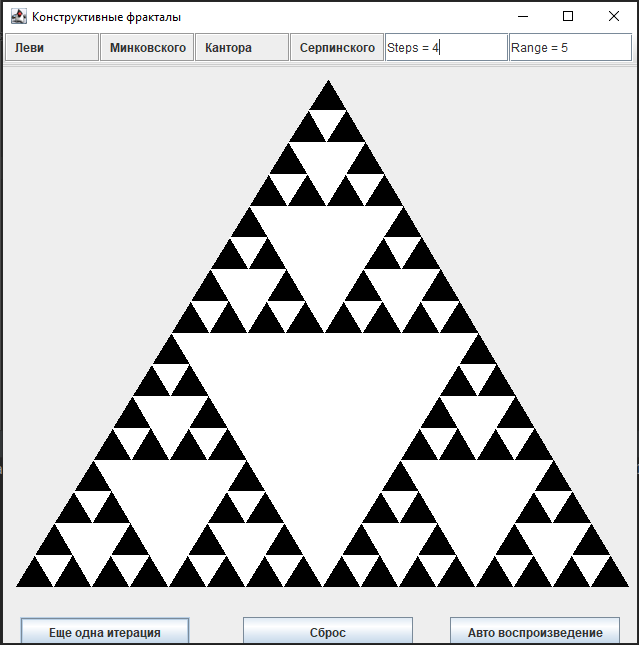


Рисунок 2 – Интерфейс приложения

Основной сценарий работы пользователя с приложением:

1. Пользователь видит кнопки с названиями фракталов и поля ввода позиций и диапазона рисунка геометрических фигур.
2. Выбирает нужную позицию, вводя число в поле «Step», диапазон, вводя число в поле «Range».
3. Нажимает на одно из названий фракталов для последовательного построения фрактала. Далее, появляется выбранный фрактал, нарисованный на конкретной позиции.
4. При необходимости привести геометрическую фигуру в начальное положение пользователь нажимает кнопку «Сброс».
5. При необходимости автоматического построения фрактала пользователь нажимает на кнопку «Автовоспроизведение»
6. При желании самостоятельно построить следующее отображение фрактала пользователь нажимает на кнопку «Еще одна итерация»
7. 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 с использованием графического интерфейса Java Swing. Этот метод позволяет обеспечить разделение данных, управление доступом к ним, а также расширяемость и модифицируемость. Графический интерфейс позволяет удобно пользоваться приложением и экономить время.

В проекте удалось реализовать все поставленные задачи:

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.

4) Обеспечить отображение фрактала в нужной позиции.

5) Обеспечить диапазон шагов рисунка фрактала.

6) Обеспечить выбор фрактала в окне.

7) Создать удобный и понятный графический интерфейс;

8) Автоматическое отображение следующей позиции геометрической фигуры

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 javax.swing.event.DocumentEvent;

import javax.swing.event.DocumentListener;

import java.awt.\*;

import java.util.concurrent.atomic.AtomicBoolean;

public class FractalsFrame extends JFrame {

private final static int sizeOfFrame = 650;

private final AtomicBoolean key = new AtomicBoolean(false);

private int steps = 4;

private int range = 5;

private void CheckRange(){ // Проверка шагов и диапазона

if (range < 2) {range = 5;}

if (steps < 0) {steps = 0;}

if (steps >= range) {

steps = range;

}

}

FractalsFrame() {

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

//Фрактал Леви

JMenuItem LevyFractalItem = new JMenuItem("Леви");

LevyFractalItem.setSize(this.getWidth(), this.getHeight());

LevyFractalItem.addActionListener(e -> {

LevyFractalPanel panel = new LevyFractalPanel();

panel.setLayout(null);

panel.MaxStep = range;

// 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();

panel.repaint();

});

// reset

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();

});

// 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);

while (key.get()) {

try {

if (steps > 0) {

Thread.sleep(10);

steps -= 1;

if (steps <= 0) {

panel.stepIncrement();

panel.flagToContinue = true;

panel.repaint();

key.set(false);

}

} else {

Thread.sleep(1000);

}

if (!key.get()) return;

panel.stepIncrement();

panel.flagToContinue = true;

panel.repaint();

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

} catch (InterruptedException ignored) {

return;

}

}

}).start());

panel.add(updateButton).setBounds(17, sizeOfFrame - 100, 170, 30);

panel.add(resetButton).setBounds(sizeOfFrame / 2 - 170/2, sizeOfFrame - 100, 170, 30);

panel.add(autoButton).setBounds(sizeOfFrame - 203, sizeOfFrame - 100, 170, 30);

getContentPane().removeAll();

getContentPane().add(panel);

pack();

DrawSteps(steps, autoButton);

});

JMenuItem minkowskiFractalItem = new JMenuItem("Минковского");

minkowskiFractalItem.addActionListener(e -> {

MinkowskiFractalPanel panel = new MinkowskiFractalPanel();

panel.setLayout(null);

panel.MaxRange = range;

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 {

if (steps > 0) {

Thread.sleep(10);

steps -= 1;

if (steps <= 0) {

panel.stepIncrement();

panel.flagToContinue = true;

panel.repaint();

key.set(false);

}

} else {

Thread.sleep(1000);

}

if (!key.get()) return;

panel.stepIncrement();

panel.flagToContinue = true;

panel.repaint();

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

} catch (InterruptedException ignored) {

return;

}

}

}).start());

panel.add(updateButton).setBounds(17, sizeOfFrame - 100, 170, 30);

panel.add(resetButton).setBounds(sizeOfFrame / 2 - 170/2, sizeOfFrame - 100, 170, 30);

panel.add(autoButton).setBounds(sizeOfFrame - 203, sizeOfFrame - 100, 170, 30);

getContentPane().removeAll();

getContentPane().add(panel);

pack();

DrawSteps(steps, autoButton);

});

JMenuItem cantorFractalItem = new JMenuItem("Кантора");

cantorFractalItem.addActionListener(e -> {

CantorFractal panel = new CantorFractal();

panel.width = sizeOfFrame - 2\*panel.paddingX;

panel.setLayout(null);

panel.step = steps;

panel.maxRange = range;

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

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

updateButton.setFocusable(false);

updateButton.addActionListener(e1 -> {

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.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();

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

panel.repaint();

if (panel.getStep() == range) {

Thread.sleep(1000);

panel.repaint();

panel.setNullStep();

}

} catch (InterruptedException ignored) {

return;

}

}

}).start());

panel.add(updateButton).setBounds(17, sizeOfFrame - 100, 170, 30);

panel.add(resetButton).setBounds(sizeOfFrame / 2 - 170/2, sizeOfFrame - 100, 170, 30);

panel.add(autoButton).setBounds(sizeOfFrame - 203, sizeOfFrame - 100, 170, 30);

getContentPane().removeAll();

getContentPane().add(panel);

pack();

});

JMenuItem serpinskiyTriangleItem = new JMenuItem("Серпинского");

serpinskiyTriangleItem.addActionListener(e -> {

SerpinskiyFractalPanel panel = new SerpinskiyFractalPanel();

panel.sizeOfPanel = sizeOfFrame;

panel.setLayout(null);

panel.MaxStep = range;

getContentPane().removeAll();

getContentPane().add(panel);

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

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

updateButton.setFocusable(false);

updateButton.addActionListener(e12 -> {

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.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 {

if (steps > 0) {

Thread.sleep(10);

steps -= 1;

if (steps <= 0) {

panel.stepIncrement();

panel.repaint();

key.set(false);

}

} else {

Thread.sleep(1000);

}

//Thread.sleep(1000);

if (!key.get()) return;

} catch (InterruptedException interruptedException) {

return;

}

panel.stepIncrement();

panel.repaint();

if (panel.getStep() == (range + 1)) panel.setNullStep();

}

}).start());

panel.add(updateButton).setBounds(17, sizeOfFrame - 100, 170, 30);

panel.add(resetButton).setBounds(sizeOfFrame / 2 - 170/2, sizeOfFrame - 100, 170, 30);

panel.add(autoButton).setBounds(sizeOfFrame - 203, sizeOfFrame - 100, 170, 30);

pack();

DrawSteps(steps, autoButton);

});

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.addSeparator();

fileMenu.add(LevyFractalItem);

fileMenu.addSeparator();

fileMenu.add(exitItem);

JToolBar tbStyle = new JToolBar();

tbStyle.setFloatable(false);

tbStyle.add(LevyFractalItem);

tbStyle.add(minkowskiFractalItem);

tbStyle.add(cantorFractalItem);

tbStyle.add(serpinskiyTriangleItem);

//================================================================

JTextField stepField = new JTextField("Steps = " + steps, 12);

stepField.getDocument().addDocumentListener(new DocumentListener() {

@Override

public void insertUpdate(DocumentEvent e) {

System.out.println("insertUpdate");

try {

steps = (int)Double.parseDouble(stepField.getText());

System.out.println(steps);

} catch (Exception exp) {}

finally {

CheckRange();

}

}

@Override

public void removeUpdate(DocumentEvent e) {

System.out.println("removeUpdate");

try {

steps = (int)Double.parseDouble(stepField.getText());

System.out.println(steps);

} catch (Exception exp) {}

finally {

CheckRange();

}

}

@Override

public void changedUpdate(DocumentEvent e) {}

});

tbStyle.add(stepField);

JTextField RangeField = new JTextField("Range = " + range, 12);

RangeField.getDocument().addDocumentListener(new DocumentListener() {

@Override

public void insertUpdate(DocumentEvent e) {

System.out.println("insertUpdate");

try {

range = (int)Double.parseDouble(RangeField.getText());

System.out.println(range);

} catch (Exception exp) {}

finally {

CheckRange();

}

}

@Override

public void removeUpdate(DocumentEvent e) {

System.out.println("removeUpdate");

try {

range = (int)Double.parseDouble(RangeField.getText());

System.out.println(range);

} catch (Exception exp) {}

finally {

CheckRange();

}

}

@Override

public void changedUpdate(DocumentEvent e) {}

});

tbStyle.add(RangeField);

//================================================================

JMenuBar menuBar = new JMenuBar();

menuBar.add(tbStyle);

setJMenuBar(menuBar);

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

pack();

}

public void DrawSteps(int temp, JButton button) {

if (temp > 1) {

button.doClick();

} else {return;}

}

}

**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 {

// Установка отступов и ширины начальной линии

public int width = 650; // 243

public final int paddingX = 40; //170 // (int)(width\*0.05)

public final int paddingY = 15;

public final int height = 25; // Высота блока

public int step = 0;

public int maxRange = 1;

@Override

public void paintComponent(Graphics g) { // Отрисовываем компоненты

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) {

if (this.step > maxRange) this.step--;

if (step == 0) return;

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

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

step--;

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

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

}

public void stepIncrement() {

if (step >= maxRange) {

step = maxRange;

} else {

step++;

}

}

public void setNullStep() {

step = 0;

}

public int getStep() {

return step;

}

}

**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 = 650;

private int step = 0;

public int MaxStep = 5;

public boolean flagToContinue = true;

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

public static double[][] pattern = { // Координаты точек

{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() { // Отрисовка компонентов

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, новых линий

ArrayList<Point> bufferPoints = new ArrayList<>(); // буферный лист полученных точек

// 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;

if (step > MaxStep) return;

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

}

}

@Override

protected void paintComponent(Graphics g) {

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 = 650;

public int MaxRange = 3; // Указываем начальные переменные

private int step = 0;

public boolean flagToContinue = true;

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

public static double[][] pattern = { // Указываем координаты точек

{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) { // Отрисовка первых линий

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, новых линий

ArrayList<Point> bufferPoints = new ArrayList<>(); // буферный лист полученных точек

// 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 > MaxRange) 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;

public int MaxStep = 9;

public int sizeOfPanel = 650;

private int paddingX = 12;

private int paddingY = sizeOfPanel - 130;

@Override

protected void paintComponent(Graphics g) {

super.paintComponent(g);

if (step == 0) {

g.fillPolygon(new int[]{paddingX, sizeOfPanel/2, sizeOfPanel - 2\*paddingX}, new int[]{paddingY, paddingX, paddingY}, 3);

}

draw(step, new int[]{paddingX, sizeOfPanel/2, sizeOfPanel - 2\*paddingX}, new int[]{paddingY, paddingX, paddingY}, g); // рекурсивный метод

}

private void draw(int n, int[] x, int[] y, Graphics gc) {

if (this.step > MaxStep) this.step--;

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.BLACK);

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);

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(true); // Разрешаем менять разрешение окна

});

}

}