**МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ**

**УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА»**

**(СПбГУТ)**

Факультет Инфокоммуникационных сетей и систем

Кафедра Защищенных систем связи

Дисциплина Разработка защищенных сетевых приложений

**Отчет по курсовой работе**

Разработка приложения генерации рисунка по заданным параметрам

*(тема)*

Студент:

Петроченко В. В. ИБС-23

*(Ф.И.О., № группы) (подпись)*

Преподаватель:

старший преподаватель

кафедры ЗСС

Цветков А. Ю.

*(уч. степень, уч. звание, Ф.И.О.) (подпись)*

Санкт-Петербург

2023

# Содержание

[Содержание 2](#_Toc153658912)

[Введение 3](#_Toc153658913)

[**1 Описания библиотек** 4](#_Toc153658914)

[1.1.1 Основные классы 4](#_Toc153658916)

[1.1.2 Основные методы 6](#_Toc153658917)

[2. Листинг программы и UML диаграммы 10](#_Toc153658927)

[3. Примеры и ресурсы 18](#_Toc153658930)

[4. Обработка ошибок 21](#_Toc153658931)

[**Заключение** 22](#_Toc153658932)

[**Список литературы** 23](#_Toc153658933)

# **Введение**

Объектом исследования в курсовой работе является сама программа.

Предмет исследования: разработка приложения для генерации случайного рисунка по заданным параметрам.

Цель: разработать программное обеспечение в котором пользователь сможет генерировать случайные рисунки с помощью базовых фигур на координатной плоскости.

Проблема заключается в необходимости создания надежного, легко масштабируемого и удобного в использовании приложения, которое позволит пользователям эффективно генерировать случайные изображения с помощью базовых фигур на координатной плоскости. Это приложение должно обладать функциональностью выбора количества базовых фигур, случайного размещения этих фигур на координатной плоскости, вывода значения кучности фигур на координатной плоскости.

Актуальность: эта тема представляет собой важное направление в сфере различных случайных генераций фигур. Она актуальна в контексте обучения базовым навыкам языка программирования Java.

Задачи, которые требуется решить, чтобы достигнуть поставленной цели:

1. Разработку удобного пользовательского интерфейса для взаимодействия с приложением.
2. Реализацию механизмов добавления базовых фигур на координатную плоскость.
3. Обеспечение рисования фигур на плоскости.
4. Создание функционала для постоянной случайной генерации с помощью кнопки.

# **1 Описания библиотек**

## 1.1 Swing(javax.swing)

javax.swing - это библиотека для создания графического интерфейса пользователя (GUI) в Java. Она предоставляет множество компонентов для построения пользовательского интерфейса, таких как окна, кнопки, текстовые поля, списки, панели и другие элементы.

### 1.1.1 Основные классы

**1. JFrame**: Класс, представляющий главное окно приложения. Он содержит заголовок, размеры и компоненты, размещаемые в окне.

public class DrawingProgram extends JFrame

**2. JPanel**: Этот класс представляет панель, которая может содержать другие компоненты. Он используется для организации компонентов внутри окна.

JPanel panel = new JPanel();

**3. JButton**: Кнопка, которую пользователь может нажимать для выполнения действия. Вы можете добавить слушателя событий для реагирования на щелчок кнопки.

JButton drawButton = new JButton("Рисовать фигуры");

**4. JOptionPane**: Используется для отображения диалоговых окон для ввода количества фигур.

JOptionPane.*showMessageDialog*(null, "Кучность фигур: " + averageDistance/10);

### 1.1.2 Основные методы

**1. paint(Graphics g).** Этот метод переопределен из класса JFrame и используется для отрисовки элементов на графическом контексте. В данном случае, он используется для отрисовки координатной сетки и геометрических фигур.

public void paint(Graphics g)

**2. calculateAverageDistance()**: Метод для вычисления среднего расстояния между точками различных геометрических фигур. Он используется для вывода информации о среднем расстоянии в диалоговом окне после отрисовки фигур.

private double calculateAverageDistance() {

**3. drawLine(int x1, int y1, int x2, int y2)**: Отрисовка линии

drawLine(int x1, int y1, int x2, int y2):

**4.** **drawOval(int x, int y, int width, int height)**: Отрисовка окружности

**drawOval(int x, int y, int width, int height)**

**5. drawRect(int x, int y, int width, int height):** Отрисовка прямоугольника

drawRect(int x, int y, int width, int height)

Swing предоставляет множество других классов и методов для создания богатого и интерактивного пользовательского интерфейса в Java. Эти компоненты и методы позволяют разработчикам создавать удобные и интуитивно понятные GUI для своих приложений.

1.2 AWT (java.awt)

1.2.1 Основные классы

**1. Frame**: Это класс, представляющий главное окно приложения. Он похож на JFrame из javax.swing.

Frame frame = new Frame("Заголовок окна");

**2. Panel**: Панель, которая может содержать другие компоненты (например, кнопки, текстовые поля и т.д.)

Panel panel = new Panel();

**3. Button**: Элемент интерфейса, который можно нажимать для выполнения определенного действия.

Button button = new Button("текст");

**4. Label**: Используется для отображения текста на пользовательском интерфейсе.

Label label = new Label("Текст");

**5. TextField**: Поле, в которое пользователь может вводить текст.

TextField textField = new TextField();

**6. Canvas**: Позволяет рисовать графику (например, геометрические фигуры, изображения) на панели.

Canvas canvas = new Canvas();

### 1.2.2 Основные методы

**1. setLayout(LayoutManager manager)**: Устанавливает менеджер размещения компонентов на панели.

panel.setLayout(new BorderLayout());

**2. add(Component comp)**: Добавляет компонент на контейнер (например, на панель).

panel.add(button);

**3. setSize(int width, int height)**: Устанавливает размер окна или компонента.

frame.setSize(400, 300);

**4. setVisible(boolean b)**: Устанавливает видимость компонента

frame.setVisible(true);

**5. setForeground(Color c)** и **setBackground(Color c)**: Устанавливают цвет переднего и фонового плана для компонентов соответственно.

button.setForeground(Color.RED);  
panel.setBackground(Color.WHITE);

## 1.3 java.io

java.awt.event.ActionEvent- является частью библиотеки AWT (Abstract Window Toolkit) в Java и предоставляет события для пользовательских действий, таких как нажатие кнопок, выбор элементов списка и других событий, происходящих в графическом интерфейсе пользователя.

### 1.3.1 Основные классы

**1. ActionEvent**: представляет собой событие, связанное с действием пользователя, таким как нажатие кнопки. Он является подклассом EventObject и содержит информацию о событии, такую как источник события и команда, связанная с этим событием..

public void actionPerformed(ActionEvent e) {

**2. ActionListener(...)**: Интерфейс, который реализуется для обработки событий действий. Содержит метод actionPerformed(ActionEvent e), который вызывается при возникновении события действия.

(new ActionListener() drawButton.addActionListener{

**3. AbstractAction**: Абстрактный класс, который предоставляет базовую реализацию интерфейса Action. Может быть расширен для создания объектов, представляющих команды (действия) в графическом интерфейсе.

**4. Action**: Интерфейс, представляющий действие или команду, которую можно выполнить. Он содержит метод actionPerformed(ActionEvent e), который вызывается при выполнении действия..

(new ActionListener() drawButton.addActionListener{  
 @Override  
 public void actionPerformed(ActionEvent e) {

### 1.3.2 Основные методы

**1. drawButton.addActionListener(...) устанавливает слушателя событий для кнопки drawButton.**

(new ActionListener() drawButton.addActionListener{

**2. actionPerformed(ActionEvent e) содержит код для отрисовки фигур, вызова repaint() для перерисовки компонента и вывода сообщения среднего расстояния.**

public void actionPerformed(ActionEvent e)

**3. setVisible(boolean b)**: Устанавливает видимость компонента

frame.setVisible(true);

**4. setForeground(Color c)** и **setBackground(Color c)**: Устанавливают цвет переднего и фонового плана для компонентов соответственно.

button.setForeground(Color.RED);  
panel.setBackground(Color.WHITE);

## 1.6 Интегрированная среда разработки (IDE) для языка Java IntelliJ IDEA Community Edition

IntelliJ IDEA Community Edition представляет собой мощную и популярную интегрированную среду разработки (IDE) для языка Java. Разработанная компанией JetBrains, она стала отличным инструментом для создания Java-приложений и не только. Рассмотрим ключевые особенности и преимущества IntelliJ IDEA Community Edition.

1. **Бесплатность и открытый исходный код:** IntelliJ IDEA Community Edition является бесплатной и полностью открытой версией. Это позволяет разработчикам использовать все основные функции IDE без каких-либо ограничений и затрат.
2. **Мощный редактор кода:** Редактор кода IntelliJ IDEA обладает множеством возможностей, улучшающих процесс написания кода. Это автодополнение, быстрые фиксации ошибок, интегрированный отладчик и поддержка различных языков программирования.
3. **Система управления проектами:** IntelliJ IDEA обладает гибкой системой управления проектами, что позволяет разрабатывать как малые приложения, так и сложные корпоративные проекты. Встроенная поддержка систем сборки, таких как Maven и Gradle, упрощает настройку и управление зависимостями.
4. **Интеграция с системами контроля версий:** Поддержка популярных систем контроля версий, таких как Git, SVN, Mercurial, делает работу с кодом в команде удобной и эффективной. Встроенные инструменты для работы с ветками, коммитами и слияниями улучшают процесс разработки.
5. **Анализ кода и рефакторинг:** IntelliJ IDEA предоставляет множество инструментов для анализа кода, выделения потенциальных проблем и автоматического исправления. Возможности рефакторинга позволяют легко изменять структуру кода, не нарушая его работоспособность.
6. **Поддержка разнообразных технологий и фреймворков:** IDE обладает обширной поддержкой технологий и фреймворков, таких как Spring, Java EE, Android, Kotlin и многих других. Это обеспечивает разработчикам широкие возможности выбора инструментов для своих проектов.
7. **Плагины и экосистема:** IntelliJ IDEA имеет богатую экосистему плагинов, которые позволяют расширять функциональность IDE под конкретные нужды разработчика. От поддержки других языков программирования до интеграции с внешними инструментами, плагины делают IntelliJ гибкой и настраиваемой.

IntelliJ IDEA Community Edition - это выдающаяся бесплатная IDE с широким набором функциональных возможностей. Она не только облегчает разработку на Java, но также предоставляет разработчикам инструменты для повышения производительности и качества кода. Эта IDE остается популярным выбором в сообществе разработчиков Java и beyond.

# 2. Листинг программы и UML диаграммы

## 2.1 Листинг программы и описания методов

import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
  
public class DrawingProgram extends JFrame {  
 private int numOfLines, numOfCircles, numOfRectangles, numOfTriangles, numOfParabolas, numOfTrapezoids;  
  
 public DrawingProgram() {  
 setTitle("Программа генерации случайных рисунков");  
 setSize(800, 800); // Увеличиваем размер окна  
 setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
  
 JPanel panel = new JPanel();  
 getContentPane().add(panel, BorderLayout.*SOUTH*);  
  
 JButton drawButton = new JButton("Рисовать фигуры");  
 panel.add(drawButton);  
  
 (new ActionListener() drawButton.addActionListener{  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 // Отрисовка фигур здесь  
 repaint();  
 // Вывод среднего расстояния между фигурами  
 double averageDistance = calculateAverageDistance();  
 JOptionPane.*showMessageDialog*(null, "Кучность фигур: " + averageDistance/10);  
 }  
 });  
  
 // Добавление диалоговых окон для ввода количества фигур  
 numOfLines = Integer.*parseInt*(JOptionPane.*showInputDialog*("Введите количество линий:"));  
 numOfCircles = Integer.*parseInt*(JOptionPane.*showInputDialog*("Введите количество окружностей:"));  
 numOfRectangles = Integer.*parseInt*(JOptionPane.*showInputDialog*("Введите количество прямоугольников:"));  
 numOfTriangles = Integer.*parseInt*(JOptionPane.*showInputDialog*("Введите количество треугольников:"));  
 numOfParabolas = Integer.*parseInt*(JOptionPane.*showInputDialog*("Введите количество парабол:"));  
 numOfTrapezoids = Integer.*parseInt*(JOptionPane.*showInputDialog*("Введите количество трапеций:"));  
 }  
  
 @Override  
 public void paint(Graphics g) {  
 super.paint(g);  
  
 // Рисование полупрозрачной координатной сетки  
 int centerX = getWidth() / 2;  
 int centerY = getHeight() / 2;  
  
 // Устанавливаем полупрозрачность  
 Graphics2D g2d = (Graphics2D) g.create();  
 g2d.setComposite(AlphaComposite.*getInstance*(AlphaComposite.*SRC\_OVER*, 0.5f));  
  
 // Отрисовка координатной сетки  
 g2d.setColor(Color.*GRAY*);  
 for (int i = -200; i <= 200; i += 10) {  
 int x = centerX + i \* 4; // Увеличиваем масштаб  
 g2d.drawLine(x, 0, x, getHeight());  
 }  
  
 for (int i = -200; i <= 200; i += 10) {  
 int y = centerY - i \* 4; // Увеличиваем масштаб  
 g2d.drawLine(0, y, getWidth(), y);  
 }  
  
 // Очищаем графический контекст  
 g2d.dispose();  
  
 // Рисование координатной плоскости  
 g.setColor(Color.*BLACK*);  
 g.drawLine(0, centerY, getWidth(), centerY);  
 for (int i = -200; i <= 200; i += 10) {  
 int x = centerX + i \* 4; // Увеличиваем масштаб  
 g.drawLine(x, centerY - 5, x, centerY + 5);  
 g.drawString(Integer.*toString*(i), x - 5, centerY + 20);  
 }  
  
 g.drawLine(centerX, 0, centerX, getHeight());  
 for (int i = -100; i <= 100; i += 10) {  
 int y = centerY - i \* 4; // Увеличиваем масштаб  
 g.drawLine(centerX - 5, y, centerX + 5, y);  
 g.drawString(Integer.*toString*(i), centerX + 10, y + 5);  
 }  
  
 // Отрисовка фигур  
 // Отрисовка линий  
 for (int i = 0; i < numOfLines; i++) {  
 int x1 = (int) (Math.*random*() \* 200) - 100; // От -100 до 100  
 int y1 = (int) (Math.*random*() \* 200) - 100; // От -100 до 100  
 int x2 = (int) (Math.*random*() \* 200) - 100; // От -100 до 100  
 int y2 = (int) (Math.*random*() \* 200) - 100; // От -100 до 100  
 g.drawLine(centerX + x1 \* 4, centerY - y1 \* 4, centerX + x2 \* 4, centerY - y2 \* 4);  
 }  
  
 // Отрисовка окружностей  
 for (int i = 0; i < numOfCircles; i++) {  
 int x = (int) (Math.*random*() \* 200) - 100; // От -100 до 100  
 int y = (int) (Math.*random*() \* 200) - 100; // От -100 до 100  
 g.drawOval(centerX + x \* 4, centerY - y \* 4, 50, 50);  
 }  
  
 // Отрисовка прямоугольников  
 for (int i = 0; i < numOfRectangles; i++) {  
 int x = (int) (Math.*random*() \* 200) - 100; // От -100 до 100  
 int y = (int) (Math.*random*() \* 200) - 100; // От -100 до 100  
 g.drawRect(centerX + x \* 4, centerY - y \* 4, 60, 40);  
 }  
  
 // Отрисовка треугольников  
 for (int i = 0; i < numOfTriangles; i++) {  
 int x1 = (int) (Math.*random*() \* 200) - 100; // От -100 до 100  
 int y1 = (int) (Math.*random*() \* 200) - 100; // От -100 до 100  
 int x2 = (int) (Math.*random*() \* 200) - 100; // От -100 до 100  
 int y2 = (int) (Math.*random*() \* 200) - 100; // От -100 до 100  
 int x3 = (int) (Math.*random*() \* 200) - 100; // От -100 до 100  
 int y3 = (int) (Math.*random*() \* 200) - 100; // От -100 до 100  
 int[] xPoints = {centerX + x1 \* 4, centerX + x2 \* 4, centerX + x3 \* 4};  
 int[] yPoints = {centerY - y1 \* 4, centerY - y2 \* 4, centerY - y3 \* 4};  
 g.drawPolygon(xPoints, yPoints, 3);  
 }  
  
 // Отрисовка парабол  
 for (int i = 0; i < numOfParabolas; i++) {  
 int a = (int) (Math.*random*() \* 20) - 10; // Коэффициент a для параболы  
 int x;  
 int y1, y2;  
 for (int j = -100; j <= 100; j++) {  
 x = centerX + j \* 4;  
 y1 = centerY - a \* j \* j / 16; // Увеличиваем масштаб  
 y2 = centerY - a \* (j + 1) \* (j + 1) / 16; // Увеличиваем масштаб  
 g.drawLine(x, y1, x + 4, y2); // Соединяем точки линиями  
 }  
 }  
  
 // Отрисовка трапеций  
 for (int i = 0; i < numOfTrapezoids; i++) {  
 int x1 = (int) (Math.*random*() \* 200) - 100; // От -100 до 100  
 int y1 = (int) (Math.*random*() \* 200) - 100; // От -100 до 100  
 int baseWidth = (int) (Math.*random*() \* 100) + 50; // Базовая ширина трапеции  
 int height = (int) (Math.*random*() \* 40); // Высота трапеции  
 int x2 = x1 + baseWidth;  
 int x3 = x1 + (baseWidth - height) / 2;  
 int x4 = x2 - (baseWidth - height) / 2;  
 int[] xPoints = {centerX + x1 \* 4, centerX + x2 \* 4, centerX + x4 \* 4, centerX + x3 \* 4};  
 int[] yPoints = {centerY - y1 \* 4, centerY - y1 \* 4, centerY - (y1 + height) \* 4, centerY - (y1 + height) \* 4};  
 g.drawPolygon(xPoints, yPoints, 4);  
 }  
 }  
  
 private double calculateAverageDistance() {  
 int totalPoints = numOfLines \* 2 + numOfCircles + numOfRectangles \* 2 + numOfTriangles \* 3 + numOfParabolas \* 201 + numOfTrapezoids \* 4;  
 int totalDistance = 0;  
  
 for (int i = 0; i < totalPoints; i++) {  
 int x1 = (int) (Math.*random*() \* 200) - 100; // От -100 до 100  
 int y1 = (int) (Math.*random*() \* 200) - 100; // От -100 до 100  
 int x2 = (int) (Math.*random*() \* 200) - 100; // От -100 до 100  
 int y2 = (int) (Math.*random*() \* 200) - 100; // От -100 до 100  
 totalDistance += Math.*sqrt*(Math.*pow*(x2 - x1, 2) + Math.*pow*(y2 - y1, 2));  
 }  
  
 return totalDistance / totalPoints;  
 }  
  
 public static void main(String[] args) {  
 SwingUtilities.*invokeLater*(new Runnable() {  
 @Override  
 public void run() {  
 new DrawingProgram().setVisible(true);  
 }  
 });  
 }  
}

import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;

Это блок импортов, где подключаются необходимые библиотеки и классы для работы с пользовательским интерфейсом (javax.swing), java.awt, который предоставляет базовые классы и интерфейсы для рисования графики и управления окнами в приложении Java

import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;

Эти строки импортируют пакеты и классы, связанные с обработкой событий (например, нажатие кнопки) в графическом пользовательском интерфейсе. В данной программе используются для реализации слушателя событий для кнопки.

**Блок 2: Объявление класса DrawingProgram**

public class DrawingProgram extends JFrame {

public class DrawingProgram extends JFrame: Объявление класса DrawingProgram, который расширяет класс JFrame (основной класс для создания графических окон в Swing).

**Блок 3: Переменные**

private int numOfLines, numOfCircles, numOfRectangles, numOfTriangles, numOfParabolas, numOfTrapezoids;

Объявление приватных переменных для хранения количества каждого типа фигур.

**Блок 4: Конструктор DrawingProgram()**

public DrawingProgram() {  
 setTitle("Программа генерации случайных рисунков");  
 setSize(800, 800); // Увеличиваем размер окна  
 setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);

Конструктор класса DrawingProgram, который инициализирует графическое окно.

**Блок 5: Создание панели и кнопки**

JPanel panel = new JPanel();  
getContentPane().add(panel, BorderLayout.*SOUTH*);  
  
JButton drawButton = new JButton("Рисовать фигуры");  
panel.add(drawButton);

Создание панели и кнопки для отрисовки фигур.

**Блок 6: Обработчик событий кнопки**

drawButton.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 // Отрисовка фигур здесь  
 repaint();  
 // Вывод среднего расстояния между фигурами  
 double averageDistance = calculateAverageDistance();  
 JOptionPane.*showMessageDialog*(null, "Кучность фигур: " + averageDistance/10);  
 }  
});

Добавление обработчика событий для кнопки. При нажатии на кнопку выполняется отрисовка фигур и выводится среднее расстояние между ними.

**Блок 7: Ввод количества фигур**

numOfLines = Integer.*parseInt*(JOptionPane.*showInputDialog*("Введите количество линий:"));  
numOfCircles = Integer.*parseInt*(JOptionPane.*showInputDialog*("Введите количество окружностей:"));  
numOfRectangles = Integer.*parseInt*(JOptionPane.*showInputDialog*("Введите количество прямоугольников:"));  
numOfTriangles = Integer.*parseInt*(JOptionPane.*showInputDialog*("Введите количество треугольников:"));  
numOfParabolas = Integer.*parseInt*(JOptionPane.*showInputDialog*("Введите количество парабол:"));  
numOfTrapezoids = Integer.*parseInt*(JOptionPane.*showInputDialog*("Введите количество трапеций:"));

Запрос у пользователя количества каждого типа фигур через диалоговые окна.

**Блок 8: Метод paint(Graphics g)**

public void paint(Graphics g) {  
 super.paint(g);

Переопределение метода paint для рисования графических элементов.

**Блок 9: Отрисовка полупрозрачной координатной сетки**

// Рисование полупрозрачной координатной сетки  
int centerX = getWidth() / 2;  
int centerY = getHeight() / 2;  
  
// Устанавливаем полупрозрачность  
Graphics2D g2d = (Graphics2D) g.create();  
g2d.setComposite(AlphaComposite.*getInstance*(AlphaComposite.*SRC\_OVER*, 0.5f));  
  
// Отрисовка координатной сетки  
g2d.setColor(Color.*GRAY*);  
for (int i = -200; i <= 200; i += 10) {  
 int x = centerX + i \* 4; // Увеличиваем масштаб  
 g2d.drawLine(x, 0, x, getHeight());  
}  
  
for (int i = -200; i <= 200; i += 10) {  
 int y = centerY - i \* 4; // Увеличиваем масштаб  
 g2d.drawLine(0, y, getWidth(), y);  
}  
  
// Очищаем графический контекст  
g2d.dispose();

Рисование полупрозрачной координатной сетки с использованием класса Graphics2D.

**Блок 10: Отрисовка координатных осей**

// Рисование координатной плоскости  
g.setColor(Color.*BLACK*);  
g.drawLine(0, centerY, getWidth(), centerY);  
for (int i = -200; i <= 200; i += 10) {  
 int x = centerX + i \* 4; // Увеличиваем масштаб  
 g.drawLine(x, centerY - 5, x, centerY + 5);  
 g.drawString(Integer.*toString*(i), x - 5, centerY + 20);  
}  
  
g.drawLine(centerX, 0, centerX, getHeight());  
for (int i = -100; i <= 100; i += 10) {  
 int y = centerY - i \* 4; // Увеличиваем масштаб  
 g.drawLine(centerX - 5, y, centerX + 5, y);  
 g.drawString(Integer.*toString*(i), centerX + 10, y + 5);  
}

Отрисовка координатных осей и их подписей.

**Блок 11: Отрисовка фигур**

// Отрисовка фигур  
 // Отрисовка линий  
 for (int i = 0; i < numOfLines; i++) {  
 int x1 = (int) (Math.*random*() \* 200) - 100; // От -100 до 100  
 int y1 = (int) (Math.*random*() \* 200) - 100; // От -100 до 100  
 int x2 = (int) (Math.*random*() \* 200) - 100; // От -100 до 100  
 int y2 = (int) (Math.*random*() \* 200) - 100; // От -100 до 100  
 g.drawLine(centerX + x1 \* 4, centerY - y1 \* 4, centerX + x2 \* 4, centerY - y2 \* 4);  
 }  
  
  
// Отрисовка окружностей  
 for (int i = 0; i < numOfCircles; i++) {  
 int x = (int) (Math.*random*() \* 200) - 100; // От -100 до 100  
 int y = (int) (Math.*random*() \* 200) - 100; // От -100 до 100  
 g.drawOval(centerX + x \* 4, centerY - y \* 4, 50, 50);  
 }  
  
 // Отрисовка прямоугольников  
 for (int i = 0; i < numOfRectangles; i++) {  
 int x = (int) (Math.*random*() \* 200) - 100; // От -100 до 100  
 int y = (int) (Math.*random*() \* 200) - 100; // От -100 до 100  
 g.drawRect(centerX + x \* 4, centerY - y \* 4, 60, 40);  
 }  
  
 // Отрисовка треугольников  
 for (int i = 0; i < numOfTriangles; i++) {  
 int x1 = (int) (Math.*random*() \* 200) - 100; // От -100 до 100  
 int y1 = (int) (Math.*random*() \* 200) - 100; // От -100 до 100  
 int x2 = (int) (Math.*random*() \* 200) - 100; // От -100 до 100  
 int y2 = (int) (Math.*random*() \* 200) - 100; // От -100 до 100  
 int x3 = (int) (Math.*random*() \* 200) - 100; // От -100 до 100  
 int y3 = (int) (Math.*random*() \* 200) - 100; // От -100 до 100  
 int[] xPoints = {centerX + x1 \* 4, centerX + x2 \* 4, centerX + x3 \* 4};  
 int[] yPoints = {centerY - y1 \* 4, centerY - y2 \* 4, centerY - y3 \* 4};  
 g.drawPolygon(xPoints, yPoints, 3);  
 }  
  
 // Отрисовка парабол  
 for (int i = 0; i < numOfParabolas; i++) {  
 int a = (int) (Math.*random*() \* 20) - 10; // Коэффициент a для параболы  
 int x;  
 int y1, y2;  
 for (int j = -100; j <= 100; j++) {  
 x = centerX + j \* 4;  
 y1 = centerY - a \* j \* j / 16; // Увеличиваем масштаб  
 y2 = centerY - a \* (j + 1) \* (j + 1) / 16; // Увеличиваем масштаб  
 g.drawLine(x, y1, x + 4, y2); // Соединяем точки линиями  
 }  
 }  
  
 // Отрисовка трапеций  
 for (int i = 0; i < numOfTrapezoids; i++) {  
 int x1 = (int) (Math.*random*() \* 200) - 100; // От -100 до 100  
 int y1 = (int) (Math.*random*() \* 200) - 100; // От -100 до 100  
 int baseWidth = (int) (Math.*random*() \* 100) + 50; // Базовая ширина трапеции  
 int height = (int) (Math.*random*() \* 40); // Высота трапеции  
 int x2 = x1 + baseWidth;  
 int x3 = x1 + (baseWidth - height) / 2;  
 int x4 = x2 - (baseWidth - height) / 2;  
 int[] xPoints = {centerX + x1 \* 4, centerX + x2 \* 4, centerX + x4 \* 4, centerX + x3 \* 4};  
 int[] yPoints = {centerY - y1 \* 4, centerY - y1 \* 4, centerY - (y1 + height) \* 4, centerY - (y1 + height) \* 4};  
 g.drawPolygon(xPoints, yPoints, 4);  
 }

Отрисовка линий, окружностей, прямоугольников, треугольников, парабол и трапеций в соответствии с введенными пользователем значениями.

**Блок 12: Метод calculateAverageDistance()**

private double calculateAverageDistance() {

Метод для вычисления среднего расстояния между случайными точками всех фигур.

**Блок 13: Метод main**

public static void main(String[] args) {  
 SwingUtilities.*invokeLater*(new Runnable() {  
 @Override  
 public void run() {  
 new DrawingProgram().setVisible(true);  
 }  
 });  
}

Метод main для запуска приложения. Создается экземпляр класса DrawingProgram и делается видимым.

## 2.2 UML диаграммы

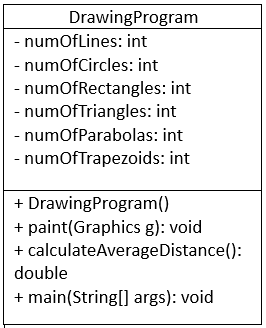


Рис. 1 - UML диаграмма классов

DrawingProgram: Главный класс программы, расширяющий JFrame. Содержит параметры для количества каждой фигуры и методы для отрисовки и вычисления среднего расстояния.

Совокупность всех возможных вариантов использования описывает функциональные возможности системы в целом.

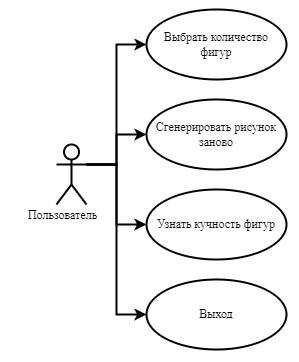


Рис.2 - Диаграмма вариантов использования

# 3. Примеры и ресурсы

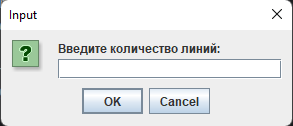


Рис.3 Интерфейс приложения

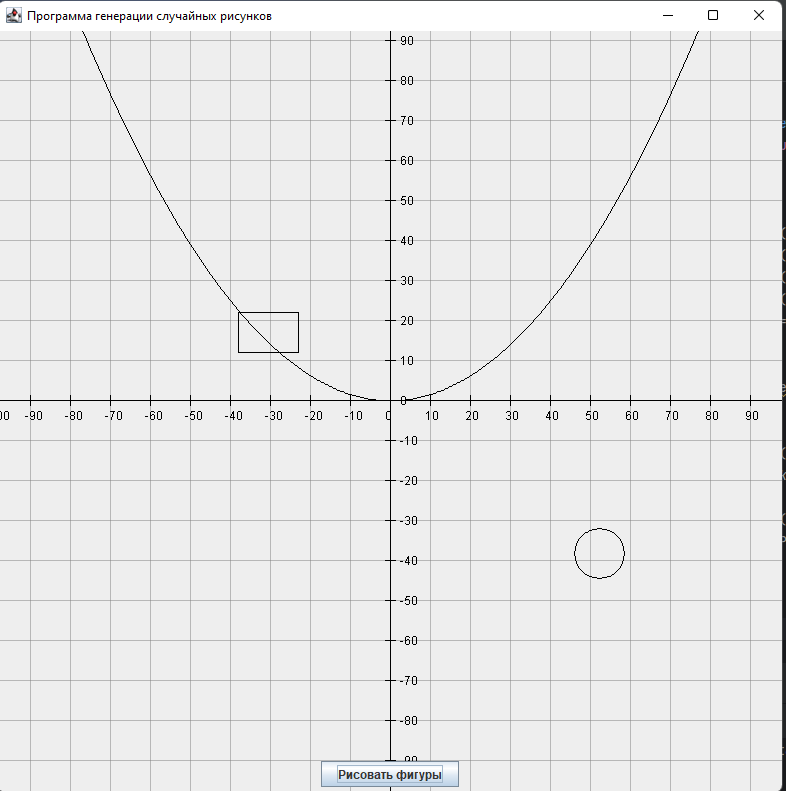


Рис.4 Успешная генерация рисунка

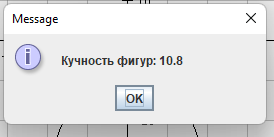


Рис.5 Вывод кучности фигур

При разработке приложения для генерации случайного изображения с использованием базовых фигур использовались различные онлайн-ресурсы для изучения технологий и инструментов. В процессе подготовки были использованы следующие источники информации:

JavaRush (javarush.ru):

На платформе JavaRush были изучены основы Java и примеры использования JavaMail API. Здесь представлены учебные материалы и квесты, включающие работу с Java swing.

Metanit (metanit.com):

Учебник по Java на сайте Metanit содержит раздел, посвященный работе с построением рисунков на координатной плоскости на Java. Были изучены примеры кода и основные концепции работы с Java API.

OracleDocumentation(https://docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.html):

Официальная документация Oracle по Java swing была использована для более глубокого понимания классов, методов.

Также были просмотрены ютуб курсы :

<https://www.youtube.com/watch?v=PJ99biaJGEQ&t=136s>

[https://www.youtube.com/watch?v=cezYoL1udqE](https://www.youtube.com/watch?v=cezYoL1udqE%20)

# 4. Обработка ошибок

В случае если пользователь введёт отрицательное число или букву программа не нарисует фигуру, а выдаст ошибку ввода с указанием причины.

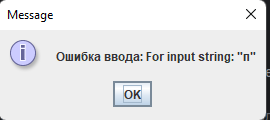
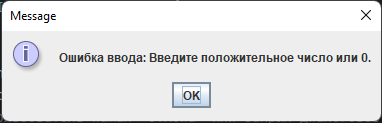


Рис.7 Вывод ошибок

Для проверки корректности ввода был использован метод getValidInput.

Этот метод использует диалоговое окно для ввода значения пользователя (JOptionPane.showInputDialog). Он затем пытается преобразовать введенное значение в целое число (Integer.parseInt). Если это значение не является положительным числом или 0, выбрасывается исключение NumberFormatException, и программа предоставляет сообщение об ошибке и повторяет ввод.

while (true) {  
 try {  
 String input = JOptionPane.*showInputDialog*(message);  
 if (input == null) {  
 // Пользователь нажал "Отмена" или закрыл диалоговое окно  
 System.*exit*(0);  
 }  
  
 int value = Integer.*parseInt*(input);  
  
 if (value < 0) {  
 throw new NumberFormatException("Введите положительное число или 0.");  
 }  
  
 return value;  
 } catch (NumberFormatException e) {  
 JOptionPane.*showMessageDialog*(null, "Ошибка ввода: " + e.getMessage());  
 }  
}

# **Заключение**

Разработка приложения для генерации случайного рисунка с помощью базовых фигур является важным шагом в обеспечении своевременного и эффективного изучения основ Java. В процессе создания данного приложения были учтены основные понятия и основы языка Java.

Использование библиотеки Swing, AWT, позволило создать функциональное приложение, способное рисовать изображения с помощью базовых фигур и вычислять из кучность.

Изучение материалов и руководств по swing на различных онлайн-платформах, таких как JavaRush, Metanit, официальная документация Oracle.

Разработанное приложение представляет собой удобный инструмент для генерирования случайных изображений, что способствует повышению эффективности и оперативности изучения языка Java.

В целом, процесс разработки данного приложения позволил расширить знания в области Java-разработки и применения swing.

# **Список литературы**

1. Шилдт, Герберт. Ш57 Java. Полное руководство, 12-е изд. : Пер. с англ. - СПб. "Диалектика•; 2023. - 1344 с.: ил. - Парал. тит. англ.

2. Блох, Джошуа Б70 Java: эффективное программирование, 3-е изд. : Пер. с англ. — СПб. : ООО “Диалектика”, 2019. — 464 с.: ил. — Парал. тит. англ.

3. Дубаков А.А. Введение в объектно-ориентированное программирование на Java: учебное пособие – СПб: Университет ИТМО, 2016. – 250 с.

4. Кэти Сьерра, Берт Бейтс **"Изучаем Java"** – 717 c.

5. Шелухин О.И., Канаев С.Д. Стеганография. Алгоритмы и программная реализация. - Горячая линия – Телеком, 2017. - 592 стр. с.

6. JDK 17 Documentation // Oracle URL: https://docs.oracle.com/en/java/javase