Київський національний університет імені Тараса Шевченка
Факультет комп’ютерних наук та кібернетики
Кафедра теорії та технології програмування

ЛАБОРАТОРНА РОБОТА №1

студент 4 курсу
групи ТТП-42
Ковалик Матвій

Науковий керівник:
Ткаченко Олексій
Миколайович

Київ-2024

Зміст

Опис завдання

Метою роботи є формування базових знань та умінь розробляти для мобільного пристрою програми обчислювального характеру, використовувати програмні засоби роботи з файлами та візуалізувати дані.

Геометричні фігури на площині:

- Вибір фігури, введення координат вершин
- Виведення значень площі, периметру
- Збереження введених даних у файл
- Креслення фігур на координатній площині
- Трикутник, чотирикутник, коло, еліпс (орієнтація вздовж осей)

Використана теорія

В даному завданні користувач вибирає геометричну фігуру (трикутник, чотирикутник, коло або еліпс) та вводить координати її вершин. Координати можуть бути представлені у вигляді пар значень (x, y). Для різних фігур координати можуть бути використані по-різному.

- Трикутник: три пари координат (x1, y1), (x2, y2), (x3, y3).

- Чотирикутник: чотири пари координат (x1, y1), (x2, y2), (x3, y3), (x4, y4).

- Коло: координати центру (x, y) і радіус r.

- Еліпс: координати центру (x, y), а також велика (a) та мала (b) півосі.

Обчислення площі та периметру для різних фігур можна реалізувати за допомогою таких формул:

$$A = \frac{1}{2} |x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2)|$$

$$P = a + b + c$$

$$P \approx \pi \left(3(a + b) - \sqrt{(3a + b)(a + 3b)}\right)$$

$$A = \pi \times a \times b$$

$$A = \pi r^2$$

$$P = 2\pi r$$

Для збереження введених даних (фігура, координати) у файл у Android можна використовувати “FileOutputStream” у режимі “MODE_PRIVATE”. Це дозволяє зберегти дані у текстовому форматі, що потім можна прочитати за допомогою “FileInputStream”.

Для графічного відображення фігур на координатній площині в Android можна використовувати клас “Canvas” у поєднанні з “Paint”. Всі фігури можна малювати, реалізуючи метод `onDraw()` у класі, що розширює “View”.

Опис алгоритмів

1. Клас Circle (Коло)

Конструктор приймає масив рядків, який містить координати центру кола та радіус. Він перевіряє, чи правильна кількість переданих параметрів. Метод для обчислення площі використовує радіус кола. Площа кола розраховується за допомогою стандартної формули, і результат округлюється до двох знаків після коми. Метод для обчислення периметру використовує радіус, щоб обчислити довжину кола.

2. Клас Ellipse (Еліпс)

Конструктор приймає координати центру еліпса та значення його півосей. Він перевіряє правильність введених даних. Метод для обчислення площі еліпса враховує значення півосей. Результат також округлюється до двох знаків після коми.Метод для обчислення периметру використовує приближену формулу, що базується на значеннях півосей еліпса. Результат округлюється до двох знаків.

3. Клас Quadrilateral (Чотирикутник)

Конструктор приймає координати чотирьох вершин. Він перевіряє правильність кількості введених точок. Метод для обчислення площі використовує координати всіх чотирьох вершин, щоб розрахувати площу чотирикутника. Результат округлюється до двох знаків. Метод для обчислення периметру сумує відстані між усіма сусідніми вершинами чотирикутника.

4. Клас Triangle (Трикутник)

Конструктор приймає координати трьох вершин трикутника. Він перевіряє правильність введення. Метод для обчислення площі використовує координати вершин, щоб визначити площу трикутника. Результат округлюється. Метод для обчислення периметру сумує довжини всіх трьох сторін трикутника, визначаючи відстані між координатами вершин.

Посібник користувача

1. Технічні вимоги

Процесор: Мінімум 1 ГГц двоядерний процесор або еквівалент.

Оперативна пам'ять (RAM): Мінімум 8 ГБ RAM. Рекомендується 16 ГБ для кращої продуктивності.

Вільний обсяг пам'яті: Мінімум 100 МБ вільного місця для встановлення застосунку та зберігання даних.

Екран: Мінімальна роздільна здатність екрана: 800x480 пікселів. Рекомендується 1280x720 або вище для кращого інтерфейсу користувача.

2. Програмні вимоги

Операційна система: Android версії 5.0 (Lollipop) або новіша. Рекомендується використовувати актуальні версії Android для кращої підтримки функцій та безпеки.

Java Development Kit (JDK): JDK версії 8 або новішої. Необхідний для компіляції коду на Java.

Android Studio: Остання версія Android Studio. Це основне середовище розробки для Android, яке включає всі необхідні інструменти для створення, налагодження та тестування застосунків.

Android SDK: Android Software Development Kit, який включає бібліотеки, інструменти та API для розробки Android-застосунків.

Дозволи: Застосунок повинен мати доступ до зовнішньої пам'яті для зберігання та читання даних файлів (якщо використовуються файли для збереження інформації).

На рисунку 1 зображено головний екран, де ми обираємо фігуру, вводимо координати обраної фігури (x1,y1;x2,y2;… для трикутника та чотирикутника, натомість для кола та еліпса - x,y,r або x,y,a,b). Натискаємо кнопку для обрахування площі та периметру. Натискаємо відповідну кнопку для збереження або читання з файлу. Для відображення інформації про автора натискаємо “Author”(Рисунок 2). Для відображення графіку натискаємо відповідну кнопку (Рисунок 3).

Рисунок1

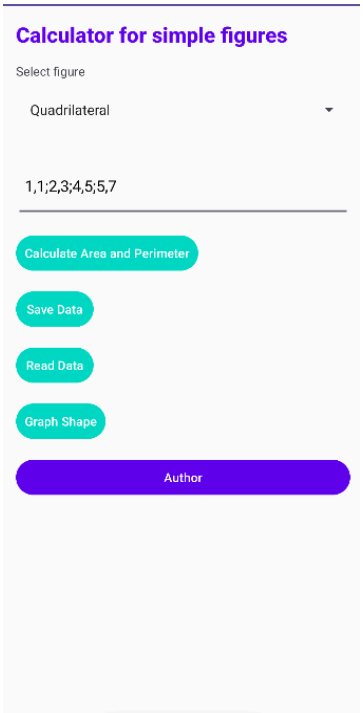


Рисунок2

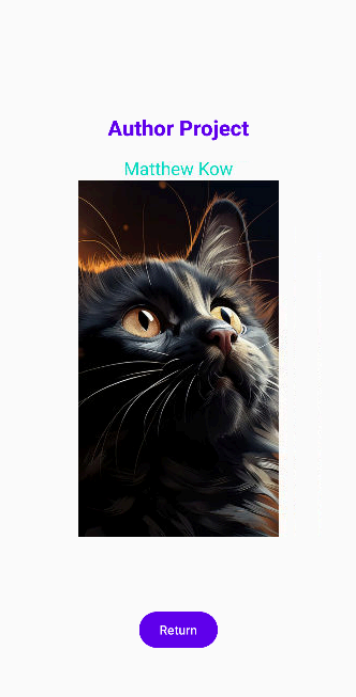
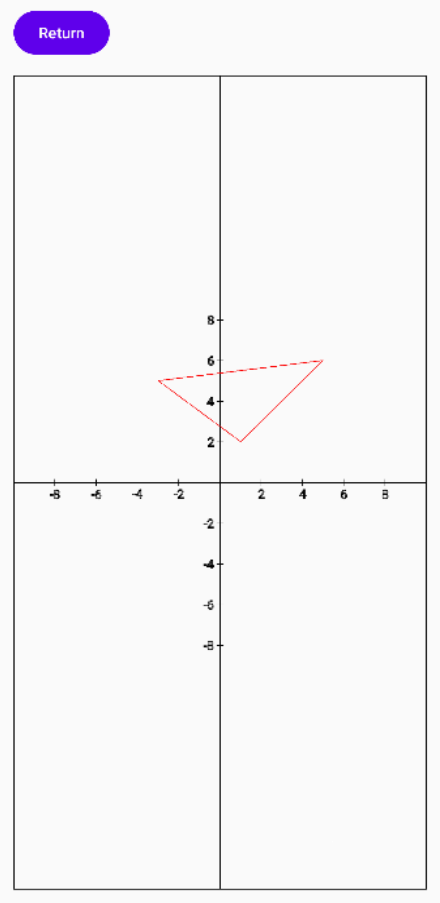


Рисунок3



Список використаних джерел

1. GeeksforGeeks. Geometric Algorithms - GeeksforGeeks. GeeksforGeeks. URL: https://www.geeksforgeeks.org/geometric-algorithms/ (date of access: 01.10.2024).
2. Canvas | Android Developers. Android Developers. URL: https://developer.android.com/reference/android/graphics/Canvas (date of access: 01.10.2024).

Додаток А

Класи для задання фігур

```
public class Circle implements Shape {
   public double radius;
   public double centerX;
   public double centerY;
   public Circle(String[] points) {
       if (points.length != 1) {
           throw new IllegalArgumentException("A circle requires 1 point (center) and a
radius.");
       }
       try {
           String[] coords = points[0].split(",");
           if (coords.length != 3) {
               throw new IllegalArgumentException("3 coordinates should be provided");
           }
           centerX = Double.parseDouble(coords[0]);
           centerY = Double.parseDouble(coords[1]);
           this.radius = Double.parseDouble(coords[2]);
       }
       catch (Exception e) {
           throw new RuntimeException("Error: " + e.getMessage());
       }
   }

   @Override
   public double calculateArea() {
       return Math.round(Math.PI * Math.pow(radius, 2) * 100.0) / 100.0;
   }

   @Override
   public double calculatePerimeter() {
       return Math.round(2 * Math.PI * radius * 100.0) / 100.0;
   }
}
```

```
public class Ellipse implements Shape {
   public double semiMajorAxis;
   public double semiMinorAxis;

   public double x;
   public double y;

   public Ellipse(String[] points) {
       if (points.length != 1) {
           throw new IllegalArgumentException("An ellipse requires 1 point (center) and two
axes.");
       }
       try {
           String[] coords = points[0].split(",");
           if (coords.length != 4) {
               throw new IllegalArgumentException("4 values should be provided - x, y, a, b");
           }
           x = Double.parseDouble(coords[0]);
           y = Double.parseDouble(coords[1]);
           semiMajorAxis = Double.parseDouble(coords[2]);
           semiMinorAxis = Double.parseDouble(coords[3]);
       } catch (Exception e) {
           throw new RuntimeException("Error: " + e.getMessage());
       }
   }

   @Override
   public double calculateArea() {
       return Math.round(Math.PI * semiMajorAxis * semiMinorAxis * 100.0) / 100.0;
```

```
    }

    @Override
    public double calculatePerimeter() {
        return Math.round((Math.PI * (3 * (semiMajorAxis + semiMinorAxis) -
                Math.sqrt((3 * semiMajorAxis + semiMinorAxis) * (semiMajorAxis + 3 *
semiMinorAxis)))) * 100.0) / 100.0;
    }
}
```

```
public class Quadrilateral implements Shape {
    public double[] x, y;

    public Quadrilateral(String[] points) {
        if (points.length != 4) {
            throw new IllegalArgumentException("A quadrilateral requires 4 points.");
        }
        try {
            x = new double[4];
            y = new double[4];
            for (int i = 0; i < 4; i++) {
                String[] coords = points[i].split(",");
                x[i] = Double.parseDouble(coords[0]);
                y[i] = Double.parseDouble(coords[1]);
            }
        } catch (Exception e)
        {
            throw new RuntimeException("Error: " + e.getMessage());
        }
    }

    @Override
    public double calculateArea() {
        return Math.round(Math.abs(((x[0] * y[1] + x[1] * y[2] + x[2] * y[3] + x[3] * y[0]) -
                (y[0] * x[1] + y[1] * x[2] + y[2] * x[3] + y[3] * x[0])) / 2.0) * 100.0) /
100.0;
    }

    @Override
    public double calculatePerimeter() {
        return Math.round((distance(x[0], y[0], x[1], y[1]) +
                distance(x[1], y[1], x[2], y[2]) +
                distance(x[2], y[2], x[3], y[3]) +
                distance(x[3], y[3], x[0], y[0])) * 100.0) / 100.0;
    }

    private double distance(double x1, double y1, double x2, double y2) {
        return Math.sqrt(Math.pow(x2 - x1, 2) + Math.pow(y2 - y1, 2));
```

```
public class Triangle implements Shape {
    public double[] x, y;

    public Triangle(String[] points) {
        if (points.length != 3) {
            throw new IllegalArgumentException("A triangle requires 3 points.");
        }
        try {
            x = new double[3];
            y = new double[3];
            for (int i = 0; i < 3; i++) {
                String[] coords = points[i].split(",");
                x[i] = Double.parseDouble(coords[0]);
                y[i] = Double.parseDouble(coords[1]);
            }
        } catch (Exception e) {
            throw new RuntimeException("Error: " + e.getMessage());
        }
```

```
    }

    public double[] getX() {
        return x;
    }

    public double[] getY() {
        return y;
    }

    @Override
    public double calculateArea() {
        return Math.round(Math.abs((double) ((x[0] * (y[1] - y[2]) + x[1] * (y[2] - y[0]) +
x[2] * (y[0] - y[1]))) / 2.0) * 100.0) / 100.0;
    }

    @Override
    public double calculatePerimeter() {
        return Math.round((distance(x[0], y[0], x[1], y[1]) +
                distance(x[1], y[1], x[2], y[2]) +
                distance(x[2], y[2], x[0], y[0])) * 100.0) / 100.0;
    }

    private double distance(double x1, double y1, double x2, double y2) {
        return Math.sqrt(Math.pow(x2 - x1, 2) + Math.pow(y2 - y1, 2));
    }
}
```

```
public interface Shape {
    double calculateArea();
    double calculatePerimeter();}
```

Класи активностей для обробки вікон

```
public class AuthorActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_author);
    }

    public void openMainPage(View view) {
        Intent intent = new Intent(this, MainActivity.class);
        startActivity(intent);
    }
}
```

```
public class GraphActivity extends AppCompatActivity {

    private String shapeType;
    private String[] points;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_graph);
        try {
            shapeType = getIntent().getStringExtra("shapeType");
            String coordinates = getIntent().getStringExtra("coordinates");
            points = coordinates.split(";");

            if (shapeType == null || points.length == 0) {
                throw new IllegalArgumentException("Error in arguments");
            }

            switch (shapeType) {
```

```
                case "Circle":
                    drawCircle();
                    break;
                case "Ellipse":
                    drawEllipse();
                    break;
                case "Triangle":
                    drawTriangle();
                    break;
                case "Quadrilateral":
                    drawQuadrilateral();
                    break;
                default:
                    break;
            }
        } catch (Exception e) {
            Toast.makeText(this, "Error drawing shape: " + e.getMessage(),
Toast.LENGTH_LONG).show();
        }

    }

    private void drawCircle() {
        Circle circle = new Circle(points);
        float radius = (float) circle.radius;
        double centerX = circle.centerX;
        double centerY = circle.centerY;
        Graph graph = new Graph.Builder()

                .addFunction(x -> Math.sqrt(Math.pow(radius, 2) - Math.pow(x - centerX, 2)) +
centerY, Color.RED)
                .addFunction(x -> -Math.sqrt(Math.pow(radius, 2) - Math.pow(x - centerX, 2)) +
centerY, Color.RED)
                .setWorldCoordinates(-10, 10, -20, 20)
                .build();
        GraphView graphView = findViewById(R.id.graph_view);
        graphView.setGraph(graph);
    }

    private void drawEllipse() {
        Ellipse ellipse = new Ellipse(points);
        double centerX = ellipse.x;
        double centerY = ellipse.y;
        double semiMajor = ellipse.semiMajorAxis;
        double semiMinor = ellipse.semiMinorAxis;

        Graph graph = new Graph.Builder()
                .addFunction(x -> centerY + semiMinor * Math.sqrt(1 - Math.pow((x - centerX) /
semiMajor, 2)), Color.RED)
                .addFunction(x -> centerY - semiMinor * Math.sqrt(1 - Math.pow((x - centerX) /
semiMajor, 2)), Color.RED)
                .setWorldCoordinates(-10, 10, -20, 20)
                .build();

        GraphView graphView = findViewById(R.id.graph_view);
        graphView.setGraph(graph);
    }

    private void drawTriangle() {
        Triangle triangle = new Triangle(points);


        double x1 = triangle.x[0];
        double y1 = triangle.y[0];
        double x2 = triangle.x[1];
        double y2 = triangle.y[1];
        double x3 = triangle.x[2];
```

```
        double y3 = triangle.y[2];


        double m1 = (y2 - y1) / (x2 - x1);
        double b1 = y1 - m1 * x1;

        double m2 = (y3 - y2) / (x3 - x2);
        double b2 = y2 - m2 * x2;

        double m3 = (y1 - y3) / (x1 - x3);
        double b3 = y3 - m3 * x3;

        Graph graph = new Graph.Builder()
                .addFunction(x -> (x >= Math.min(x1, x2) && x <= Math.max(x1, x2)) ? m1 * x +
b1 : 1000, Color.RED)
                .addFunction(x -> (x >= Math.min(x2, x3) && x <= Math.max(x2, x3)) ? m2 * x +
b2 : 1000, Color.RED)
                .addFunction(x -> (x >= Math.min(x1, x3) && x <= Math.max(x1, x3)) ? m3 * x +
b3 : 1000, Color.RED)
                .setWorldCoordinates(-10, 10, -20, 20)
                .build();


        GraphView graphView = findViewById(R.id.graph_view);
        graphView.setGraph(graph);
    }

    private void drawQuadrilateral() {
        Quadrilateral quadrilateral = new Quadrilateral(points);
        double x1 = quadrilateral.x[0];
        double y1 = quadrilateral.y[0];
        double x2 = quadrilateral.x[1];
        double y2 = quadrilateral.y[1];
        double x3 = quadrilateral.x[2];
        double y3 = quadrilateral.y[2];
        double x4 = quadrilateral.x[3];
        double y4 = quadrilateral.y[3];

        double m1 = (y2 - y1) / (x2 - x1);
        double b1 = y1 - m1 * x1;

        double m2 = (y3 - y2) / (x3 - x2);
        double b2 = y2 - m2 * x2;

        double m3 = (y4 - y3) / (x4 - x3);
        double b3 = y3 - m3 * x3;

        double m4 = (y1 - y4) / (x1 - x4);
        double b4 = y4 - m4 * x4;

        Graph graph = new Graph.Builder()
                .addFunction(x -> (x >= Math.min(x1, x2) && x <= Math.max(x1, x2)) ? m1 * x +
b1 : 1000, Color.RED)
                .addFunction(x -> (x >= Math.min(x2, x3) && x <= Math.max(x2, x3)) ? m2 * x +
b2 : 1000, Color.RED)
                .addFunction(x -> (x >= Math.min(x4, x3) && x <= Math.max(x4, x3)) ? m3 * x +
b3 : 1000, Color.RED)
                .addFunction(x -> (x >= Math.min(x4, x1) && x <= Math.max(x4, x1)) ? m4 * x +
b4 : 1000, Color.RED)
                .setWorldCoordinates(-10, 10, -20, 20)
                .build();

        GraphView graphView = findViewById(R.id.graph_view);
        graphView.setGraph(graph);

    }
```

```
    public void openMainPage(View view) {
        Intent intent = new Intent(this, MainActivity.class);
        startActivity(intent);
    }
}
```

```
public class MainActivity extends AppCompatActivity {

    private Spinner shapeSpinner;
    private EditText coordinatesInput;
    private Button calculateButton;
    private Button saveButton;
    private Button graphButton;
    private Button readButton;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        shapeSpinner = findViewById(R.id.shapeSpinner);
        coordinatesInput = findViewById(R.id.coordinatesInput);
        calculateButton = findViewById(R.id.calculateButton);
        saveButton = findViewById(R.id.saveButton);
        graphButton = findViewById(R.id.graphButton);
        readButton = findViewById(R.id.readButton);

        String[] shapes = {"Triangle", "Circle", "Ellipse", "Quadrilateral"};
        ArrayAdapter<String> adapter = new ArrayAdapter<>(this,
android.R.layout.simple_spinner_item, shapes);
        adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
        shapeSpinner.setAdapter(adapter);

        calculateButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                try {
                    String shapeType = shapeSpinner.getSelectedItem().toString();
                    String[] points = coordinatesInput.getText().toString().split(";");
                    var shape = ShapeFactory.createShape(shapeType, points);
                    double area = shape.calculateArea();
                    double perimeter = shape.calculatePerimeter();
                    Toast.makeText(MainActivity.this,
                            "Area: " + area + ", Perimeter: " + perimeter,
                            Toast.LENGTH_LONG).show();
                } catch (Exception e) {
                    Toast.makeText(MainActivity.this,
                            "Error: " + e.getMessage(),
                            Toast.LENGTH_LONG).show();
                }
            }
        });

        saveButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                String shapeType = shapeSpinner.getSelectedItem().toString();
                String coordinates = coordinatesInput.getText().toString();

                String dataToSave = "Shape: " + shapeType + "\nCoordinates: " + coordinates +
"\n\n";

                try {
                    if (shapeType == null || coordinates == null || coordinates.isEmpty()) {
                        throw new IllegalArgumentException("Empty values");
                    }
```

```
                String[] points_check = coordinatesInput.getText().toString().split(";");
                var shape = ShapeFactory.createShape(shapeType, points_check);
                FileOutputStream fos = openFileOutput("data.txt", MODE_PRIVATE);
                fos.write(dataToSave.getBytes());
                fos.close();

                Toast.makeText(MainActivity.this, "Data saved to data.txt!",
Toast.LENGTH_LONG).show();
            } catch (Exception e) {
                e.printStackTrace();
                Toast.makeText(MainActivity.this, "Error saving data: " + e.getMessage(),
Toast.LENGTH_LONG).show();
            }
        }
    });

    readButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            try {
                FileInputStream fis = openFileInput("data.txt");
                InputStreamReader isr = new InputStreamReader(fis);
                BufferedReader bufferedReader = new BufferedReader(isr);
                StringBuilder stringBuilder = new StringBuilder();
                String line;
                String lastShapeType = "";
                String lastCoordinates = "";

                while ((line = bufferedReader.readLine()) != null) {
                    stringBuilder.append(line).append("\n");
                    if (line.startsWith("Shape: ")) {
                        lastShapeType = line.substring("Shape: ".length()).trim();
                    }
                    // Check if the line contains coordinates
                    if (line.startsWith("Coordinates: ")) {
                        lastCoordinates = line.substring("Coordinates: ".length()).trim();
                    }
                }

                fis.close();

                if (!lastShapeType.isEmpty()) {
                    int spinnerPosition = ((ArrayAdapter<String>)
shapeSpinner.getAdapter()).getPosition(lastShapeType);
                    if (spinnerPosition >= 0) {
                        shapeSpinner.setSelection(spinnerPosition);
                    }
                }

                coordinatesInput.setText(lastCoordinates);
                Toast.makeText(MainActivity.this, "Data read from data.txt!",
Toast.LENGTH_LONG).show();
            } catch (Exception e) {
                e.printStackTrace();
                Toast.makeText(MainActivity.this, "Error showing graph: " + e.getMessage(),
Toast.LENGTH_LONG).show();
            }
        }
    });

    graphButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            try {
                String shapeType = shapeSpinner.getSelectedItem().toString();
                String coordinates = coordinatesInput.getText().toString();
```

```
                if (shapeType == null || coordinates == null || coordinates.isEmpty()) {
                    throw new IllegalArgumentException("Empty values");
                }

                String[] points_check = coordinatesInput.getText().toString().split(";");
                var shape = ShapeFactory.createShape(shapeType, points_check);

                Intent intent = new Intent(MainActivity.this, GraphActivity.class);
                intent.putExtra("shapeType", shapeType);
                intent.putExtra("coordinates", coordinates);
                startActivity(intent);
            } catch (Exception e) {
                e.printStackTrace();
                Toast.makeText(MainActivity.this, "Error showing graph: " + e.getMessage(),
Toast.LENGTH_LONG).show();
            }
        }
    });
    Button authorButton = findViewById(R.id.Author);
    authorButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            openAuthorActivity();
        }
    });
}

public void openAuthorActivity() {
    try {
        Intent intent = new Intent(this, AuthorActivity.class);
        startActivity(intent);
    } catch (Exception e) {
        e.printStackTrace();
        Toast.makeText(MainActivity.this, "Error showing author: " + e.getMessage(),
Toast.LENGTH_LONG).show();
    }
}
}
```