КАЗАНСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ

Институт вычислительной математики и информационных технологий

Кафедра системного анализа и информационных технологий

Отчёт

по дисциплине «Программирование на Java»

**«Разработка Android приложения»**

Выполнил: студент группы 09–032

Краев Максим Максимович

Преподаватель:

Жуманиёзов Алишер Равшонбекович

Казань – 2022

**Содержание**

[Семестровая (1 часть) 3](#_Toc122745524)

[Семестровая (2 часть) 6](#_Toc122745525)

[Семестровая (3 часть) 12](#_Toc122745526)

[Семестровая (4 часть) 15](#_Toc122745527)

[Семестровая (5 часть) 17](#_Toc122745528)

[Код 21](#_Toc122745529)

[Консольное приложение 21](#_Toc122745530)

[Графическое приложение 43](#_Toc122745531)

[Android-приложение 93](#_Toc122745532)

# Семестровая (1 часть)

В первой семестровой работе необходимо было создать для каждого из типов геометрических фигур свой класс и интерфейсы. Прежде чем вообще начинать построение классов для геометрических фигур, необходимо было построить класс точки, а потом и класс точки размерности 2. Так как есть обычная точка в одномерном пространстве и двумерном, также был написан класс точки и в трехмерном пространстве.

Для каждой фигуры, были реализованы в классах функции для получения координат разного типа, реализованы геттеры и сеттеры, функция для подсчета площади, периметра, поворота, отражение относительно осей, перемещения.

Также были построены в классах функции для пересечения фигур.

Чтобы проверить правильность и корректность работы, необходимо было написать основной проект, в котором осуществлялись бы манипуляции с фигурами. Main.java – основной проект, с которого начинается работа консольного приложения.

В начале работы приложения нужно ввести количество фигур и после ввести название фигур и их параметры.

Введенные фигуры сохранялись в список ArrayList<IShape> figures.

В конце выводилась общая площадь фигур, средняя площадь и общий периметр фигур.

После вводились фигуры, соответствующие i-ому типу фигуры в списке, параметры для создания фигуры и параметры для перемещения. После ввода необходимо было вывести пересекаются ли фигуры до и после перемещения. Так продолжается до тех пор, пока количество, веденных фигур не достигнет для списка ArrayList<IShape> figures.

Входные данные во время выполнения работы:

8

circle

1 1

1

segment

0 2

1 1

polyline

3

0 3

2 1

1 1

ngon

5

1 1

2 3

4 6

3 0

1 0

tgon

1 5

2 10

3 8

qgon

5 7

9 8

10 3

2 2

rectangle

0 4

8 10

11 6

3 0

trapeze

-1 4

1 10

5 9

2 0

0 2

2

Shift

3 3

0 1

2 3

Rotate

1

4

0 1

-1 2

2 3

1 0

Symmetry

0

6

5 2

8 3

9 1

8 -2

6 -2

5 -1

Shift

-2 1

1 2

5 1

5 5

Rotate

0,6

-1 8

2 6

-2 3

-4 5

Symmetry

1

2 10

6 9

5 5

1 6

Shift

3 -2

6 0

4 3

6 6

9 1.5

Rotate

1.9

Выходные данные:

Square: 126,14

Length: 115,44

Average square: 15,77

Cross

No Cross

Cross

No Cross

Cross

No Cross

No Cross

Cross

No Cross

Cross

No Cross

Cross

Cross

Cross

No Cross

No Cross

# Семестровая (2 часть)

Во второй работе нужно было реализовать графический интерфейс для отображения всех фигур, которые были построены в первом проекте.

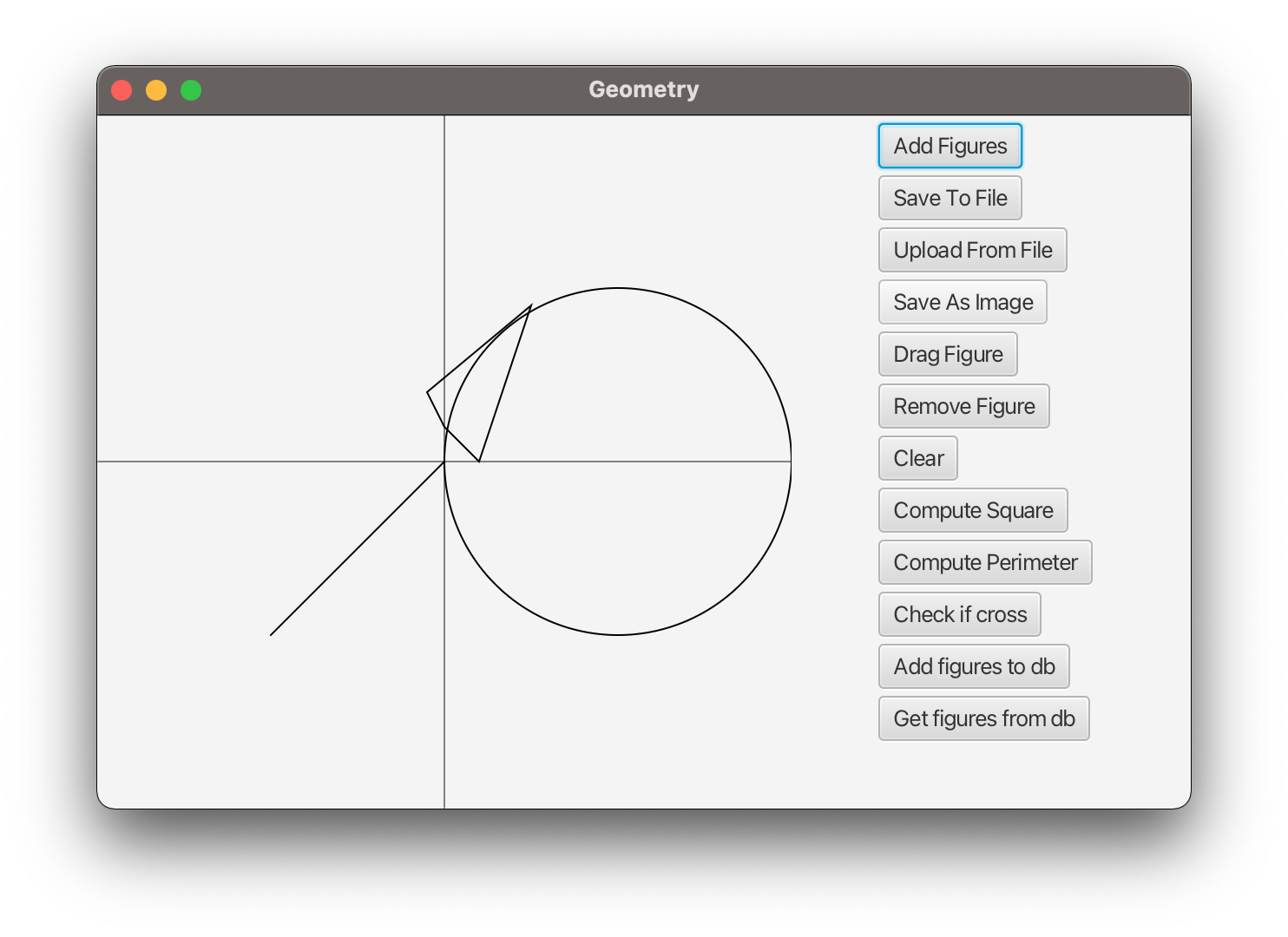
Второй проект делался совместно с третьим, был установлен gradle 7.4.2 для того, чтобы можно было подключать fxml. Все окна были реализованы с помощью fxml.

Есть возможность перехода между окнами, в которых существуют различные манипуляций с геометрическими фигурами.

Дополнительно разработаны взаимодействия с файлами: запись данных фигуры в файл, загрузка данных из файла, при этом создаются нужные классы данных, классы реализованы для того, чтобы было удобно отрисовывать фигуры и преобразовывать данные в нужный вид для программы.

В основном окне есть список фигур, который передается на другие окна при нажатии на кнопки, это нужно для того, чтобы отображать данные в новых окнах и производить изменения, потом этот список передавался в основное окно и заново отрисовывались все объекты.

Для самой отрисовки, в основном классе определялся тип фигуры и после передавался на фабрику для отрисовки. Определялся тип фигуры и исходя из типа фигуры, вызывался нужный отрисовщик. Отрисовщик в свою очередь помогал переносить изображение на canvas, в котором и были отображены все объекты.

Главный экран со случайно добавленными фигурами:

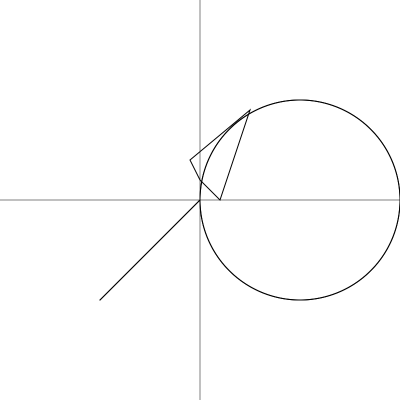
Содержимое файла, созданный после вызова функции «Save To File»:

Circle(p=Point(dims=2, x=(100.0, 0.0)), r=100,00)

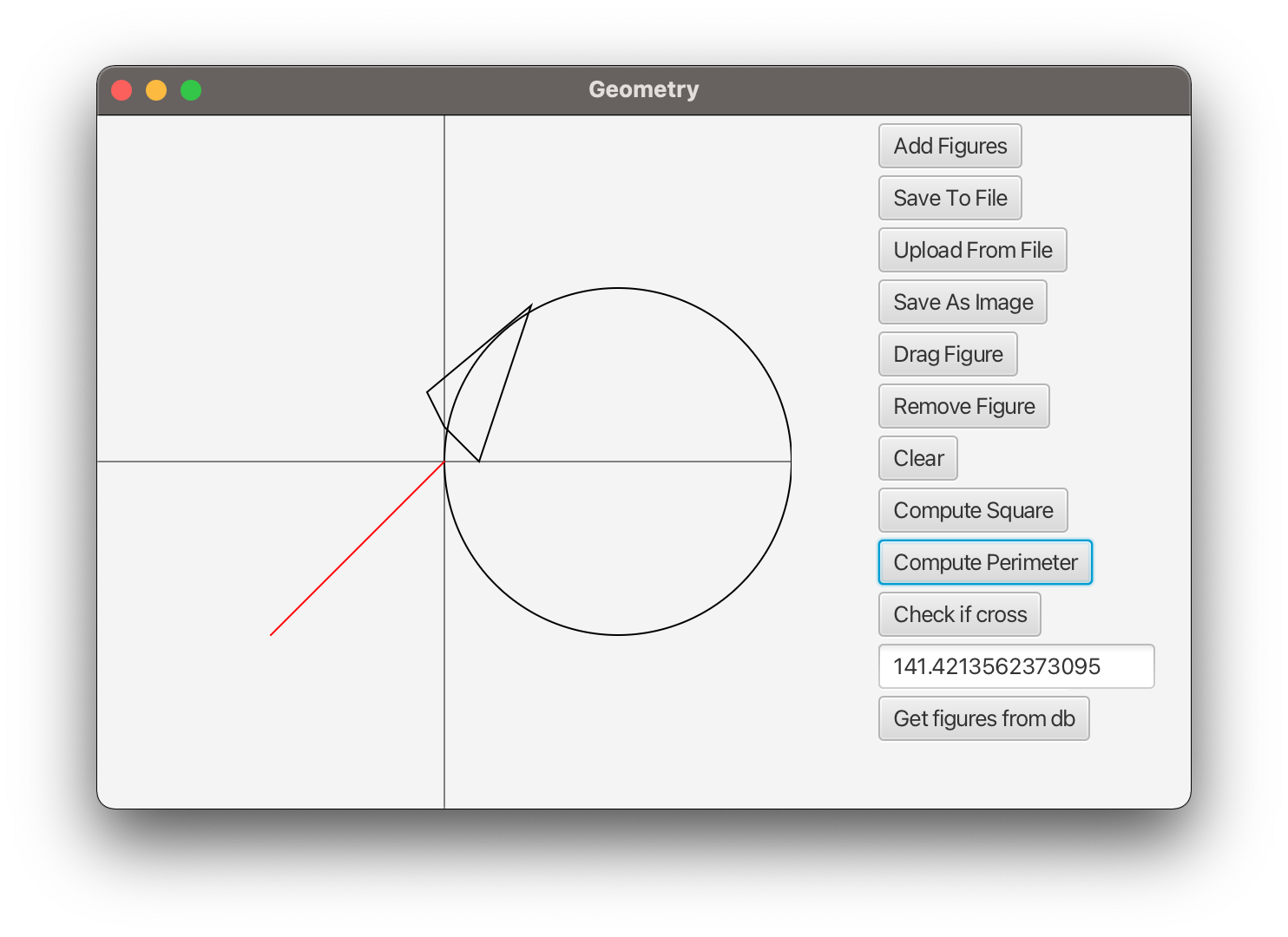
Segment(start=Point(dims=2, x=(0.0, 0.0)), finish=Point(dims=2, x=(-100.0, -100.0)))

Trapeze(n=4, p=(Point(dims=2, x=(-10.0, 40.0)), Point(dims=2, x=(50.0, 90.0)), Point(dims=2, x=(20.0, 0.0)), Point(dims=2, x=(0.0, 20.0))))

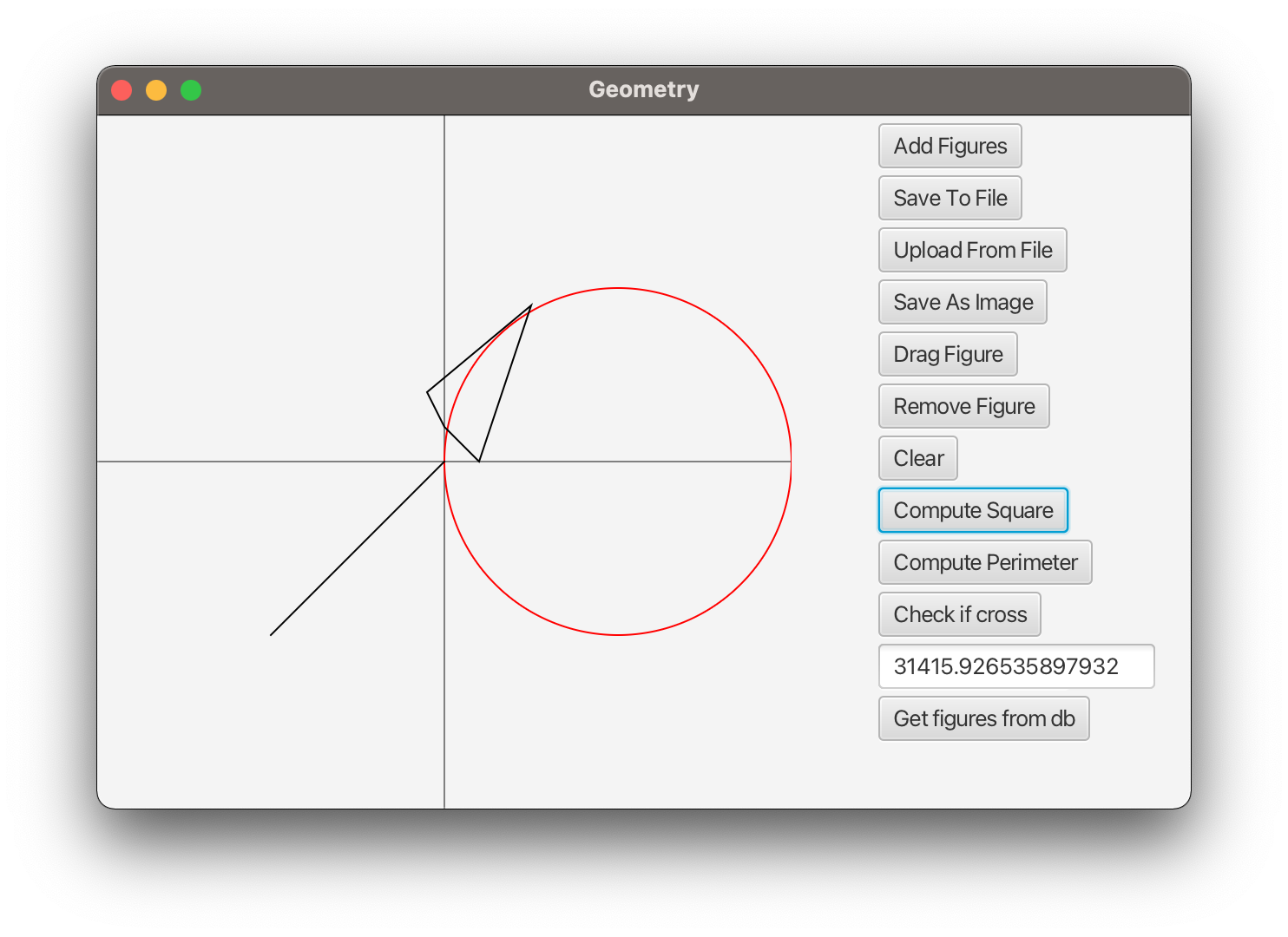
Картинка, которая создаётся после вызова функции:



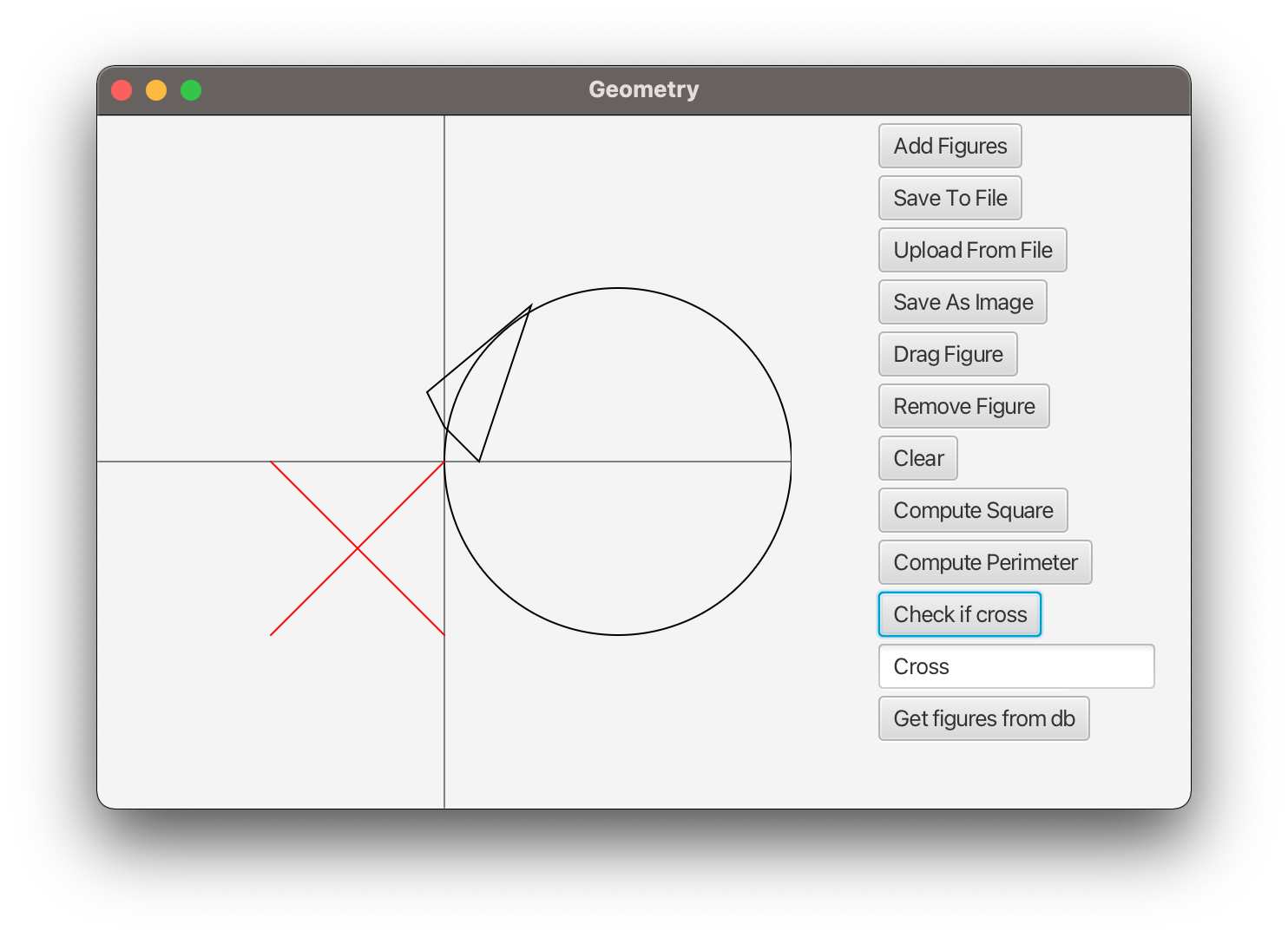
Вызов функции для нахождения длины:



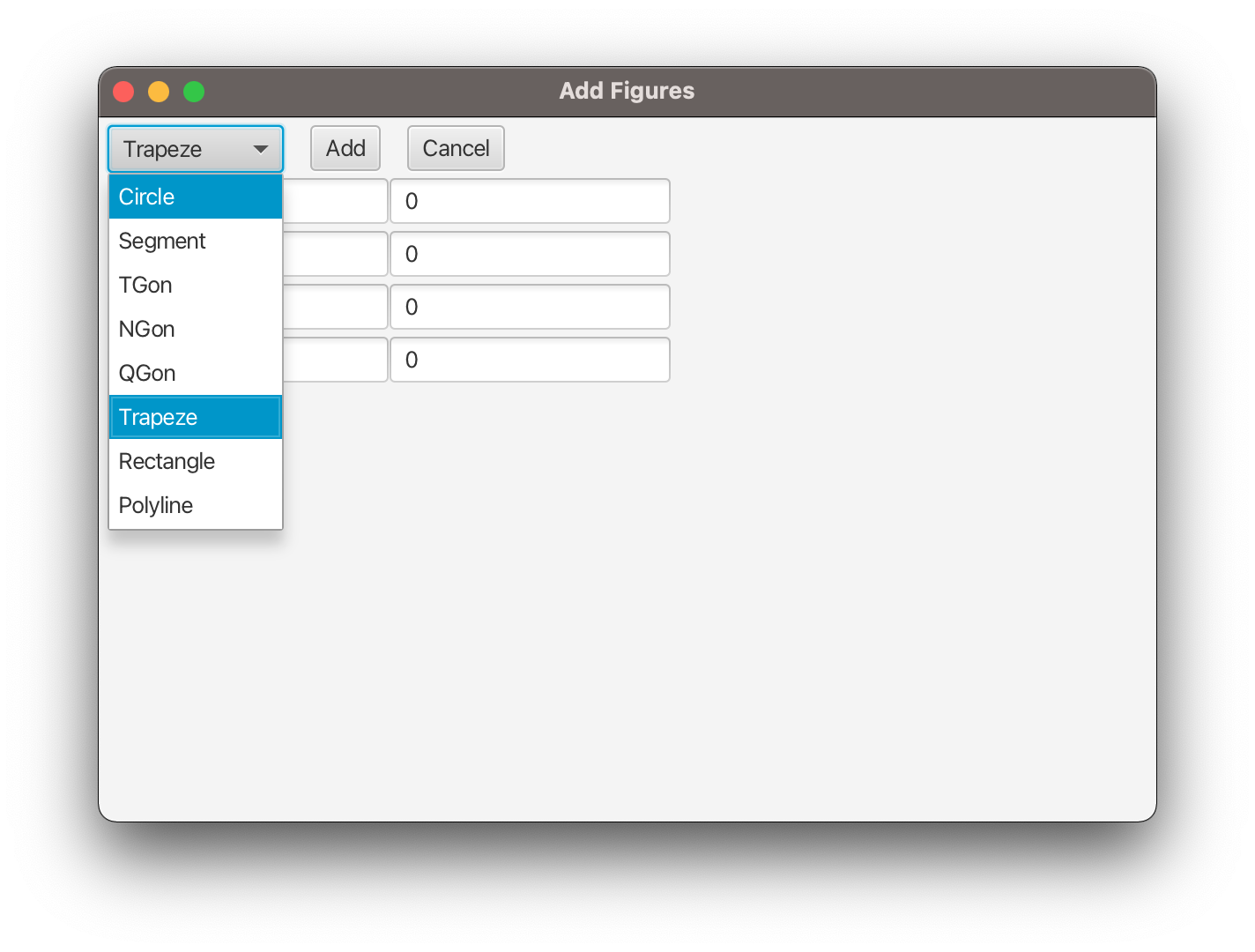
Вызов функции для нахождения площади:



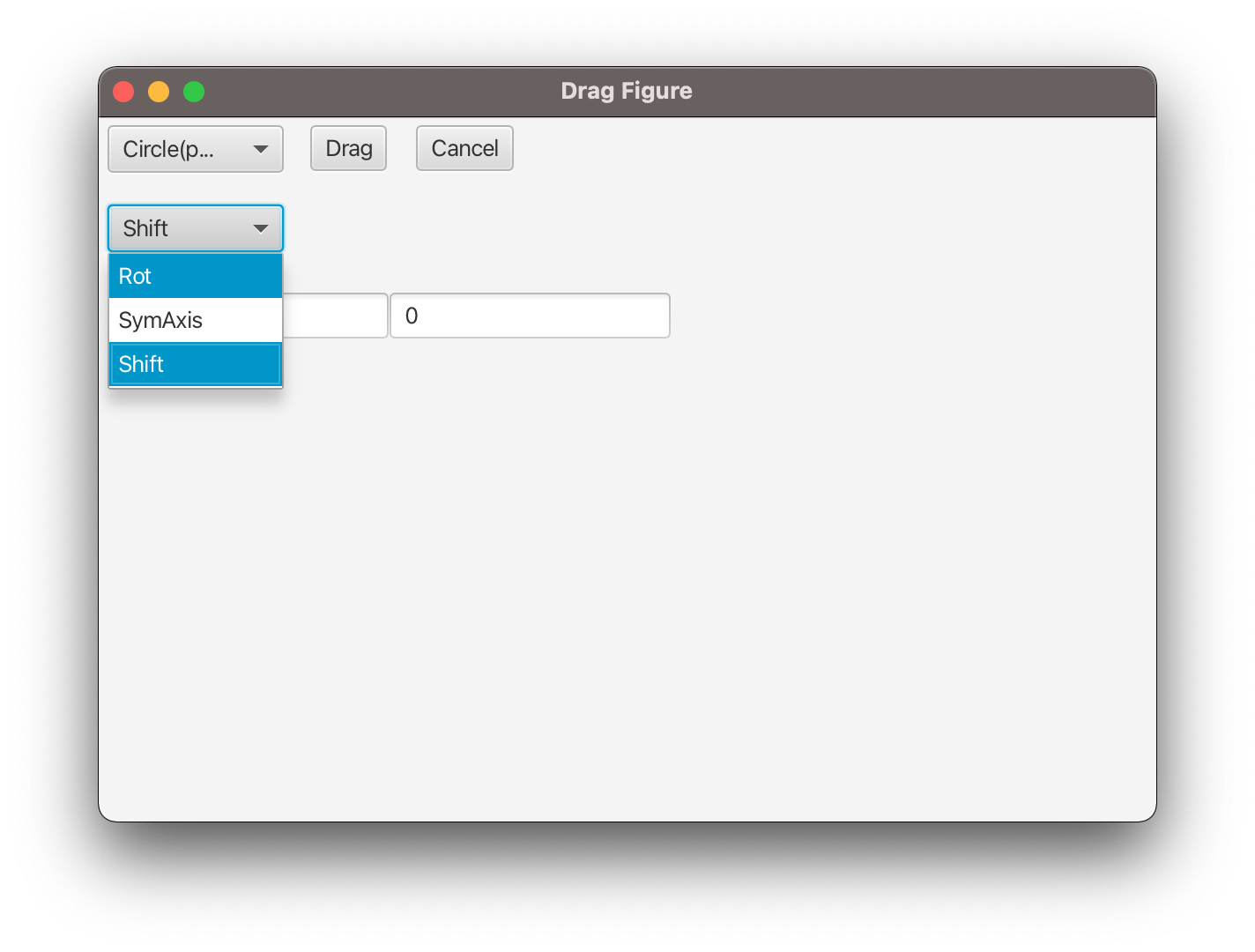
Вызов функции для проверки на пересечение:



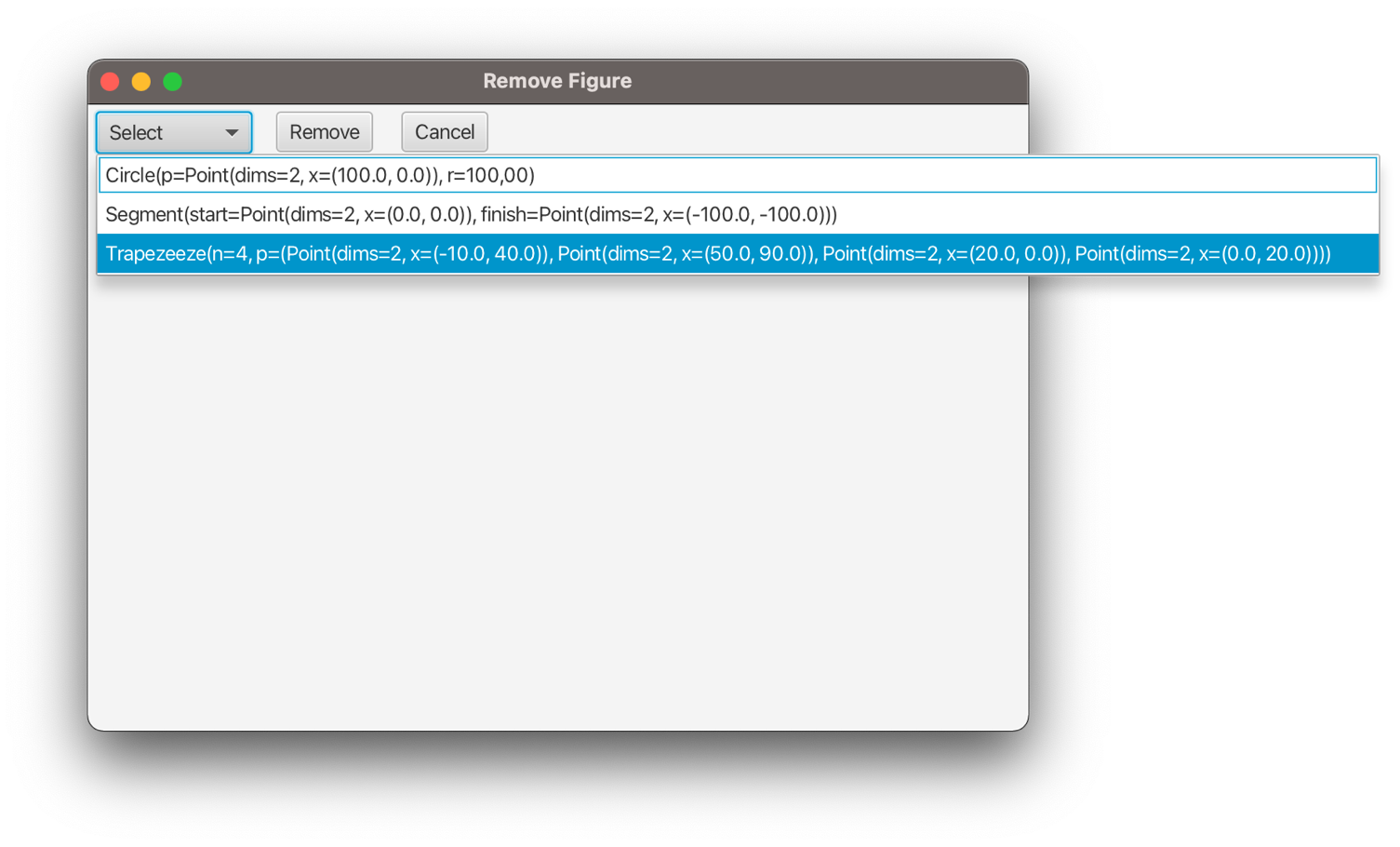
Окно для создания геометрической фигуры:



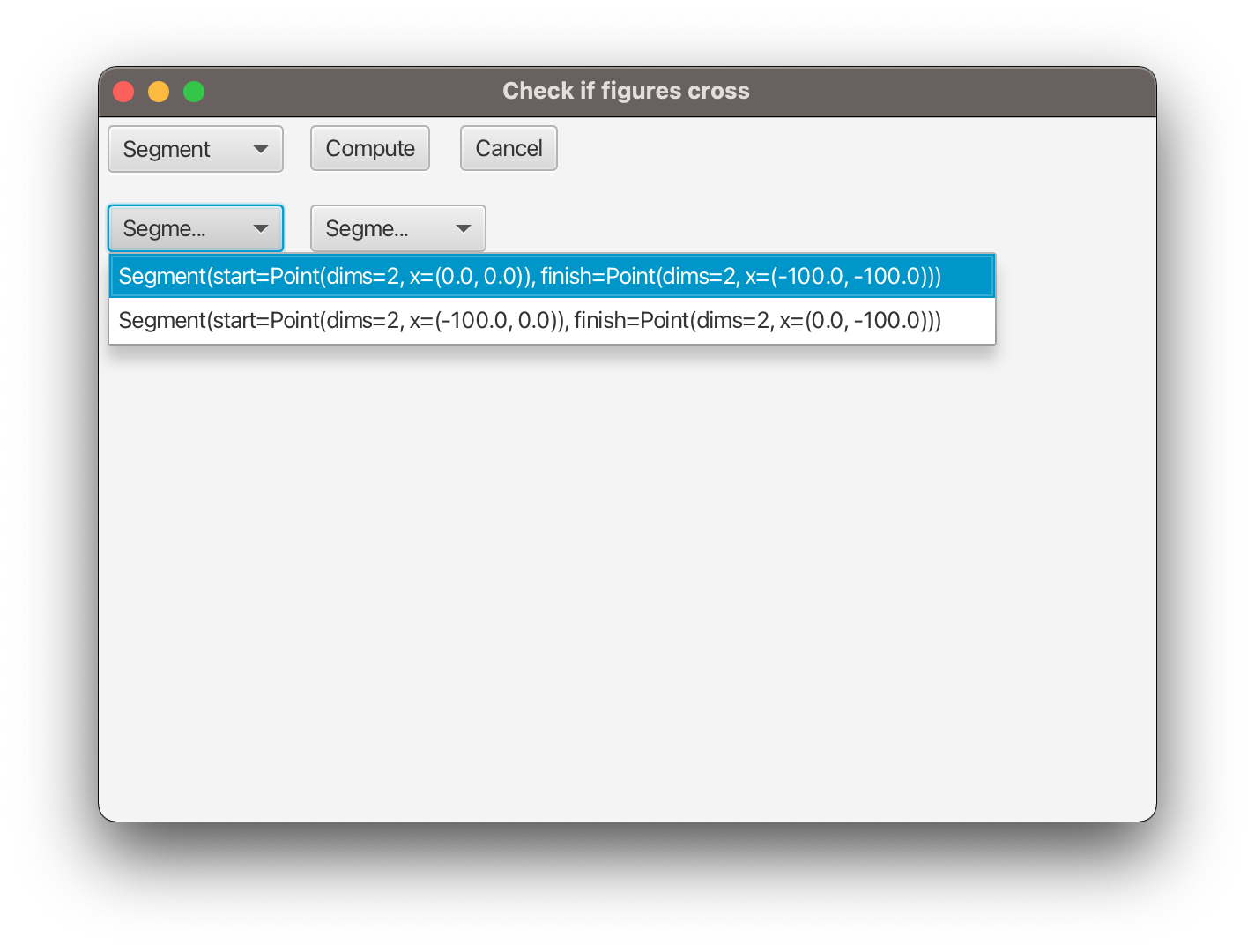
Окно для аффинного преобразования геометрической фигуры:



Окно для удаления геометрической фигуры:



Окно для проверки пересечения геометрических фигур:



# Семестровая (3 часть)

В третьем проекте необходимо было разбить проекты на несколько частей, тем самым разделить графическую часть и консольную и объединить два проекта. Для этого использовался gradle версии 7.4.2. В каждом блоке был установлен gradle, который определял для каждого блока зависимости, связи и подгружал нужные библиотеки. jar-файлы под грузились сами при внедрении gradle.

Gradle для консольного приложения:

plugins **{** id 'java-library'  
 id 'java'  
 id "io.franzbecker.gradle-lombok" version '5.0.0'  
**}**lombok **{** version = '1.18.4'  
 sha256 = ""  
**}**dependencies **{  
}**

Gradle для графического приложения:

plugins **{** id 'java'  
 id 'application'  
 id 'org.openjfx.javafxplugin' version '0.0.13'  
 id "io.franzbecker.gradle-lombok" version '5.0.0'  
**}**javafx **{** version = "18"  
 modules = [ 'javafx.controls', 'javafx.fxml', 'javafx.graphics', 'javafx.swing' ]  
**}**lombok **{** version = '1.18.4'  
 sha256 = ""  
**}**mainClassName = 'ui.Main'  
  
dependencies **{** implementation project(':data-module')  
**}**

Общий gradle для всего проекта:

subprojects **{** apply plugin: 'application'  
 apply plugin: 'java'  
 repositories **{** jcenter()  
 mavenCentral()  
 **}** dependencies **{** implementation 'org.mongodb:mongodb-driver-sync:4.1.2'  
 testImplementation 'junit:junit:4.12'  
 testRuntimeOnly("org.junit.vintage:junit-vintage-engine:5.8.2")  
 **}** application **{** mainClass = 'ui.Main'  
 **}** tasks.named('test') **{** useJUnitPlatform()   
 **}  
  
}**tasks.register('runUi') **{** dependsOn ':data-module:build'  
 dependsOn ':data-module:test'  
 dependsOn ':presentation-module:build'  
 dependsOn ':presentation-module:run'  
**}**defaultTasks 'runUi'

В общем gradle файле используются зависимости с библиотекой «junit» для проведения тестов перед запуском графического приложения. Всего было написано 23 теста на вычисление площади, периметра, проверки пересечений, перемещение фигур. Реализация тестов для NGon. Остальные реализации, предлагаю изучить в коде, который будет находиться в разделе «код».

public class NGonTest {  
 private Point2D[] points;  
 private NGon ngon;  
  
 public NGonTest() {  
 try {  
 points = new Point2D[] {  
 new Point2D(1, 1),  
 new Point2D(2, 1),  
 new Point2D(3, 0),  
 new Point2D(4, -2),  
 new Point2D(0, 0)  
 };  
 ngon = new NGon(points);  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
  
 @Test  
 public void checkLength() {  
 String testName = "ngon length";  
 try {  
 *assertTrue*(testName, Math.*abs*(ngon.length() - 10.54) < 0.01);  
 } catch (Exception e) {  
 e.printStackTrace();  
 *assertTrue*(testName, false);  
 }  
 }  
  
 @Test  
 public void checkSquare() {  
 String testName = "ngon square";  
 try {  
 *assertTrue*(testName, Math.*abs*(ngon.square() - 5) < 0.01);  
 } catch (Exception e) {  
 e.printStackTrace();  
 *assertTrue*(testName, false);  
 }  
 }  
  
 @Test  
 public void checkCross() {  
 String testName = "ngon shift";  
 try {  
 Point2D[] points1 = new Point2D[] {  
 new Point2D(-2, 1),  
 new Point2D(-3, 1),  
 new Point2D(-4, 0),  
 new Point2D(-5, -2),  
 new Point2D(-1, 0)  
 };  
 NGon ngon1 = new NGon(points1);  
 // assertTrue(testName, ngon.cross(ngon1));  
  
 double rotAngle = -Math.*PI*;  
 ngon1.rot(rotAngle);  
 *assertTrue*(testName, ngon.cross(ngon1));  
 } catch (Exception e) {  
 e.printStackTrace();  
 *assertTrue*(testName, false);  
 }  
 }  
}

# Семестровая (4 часть)

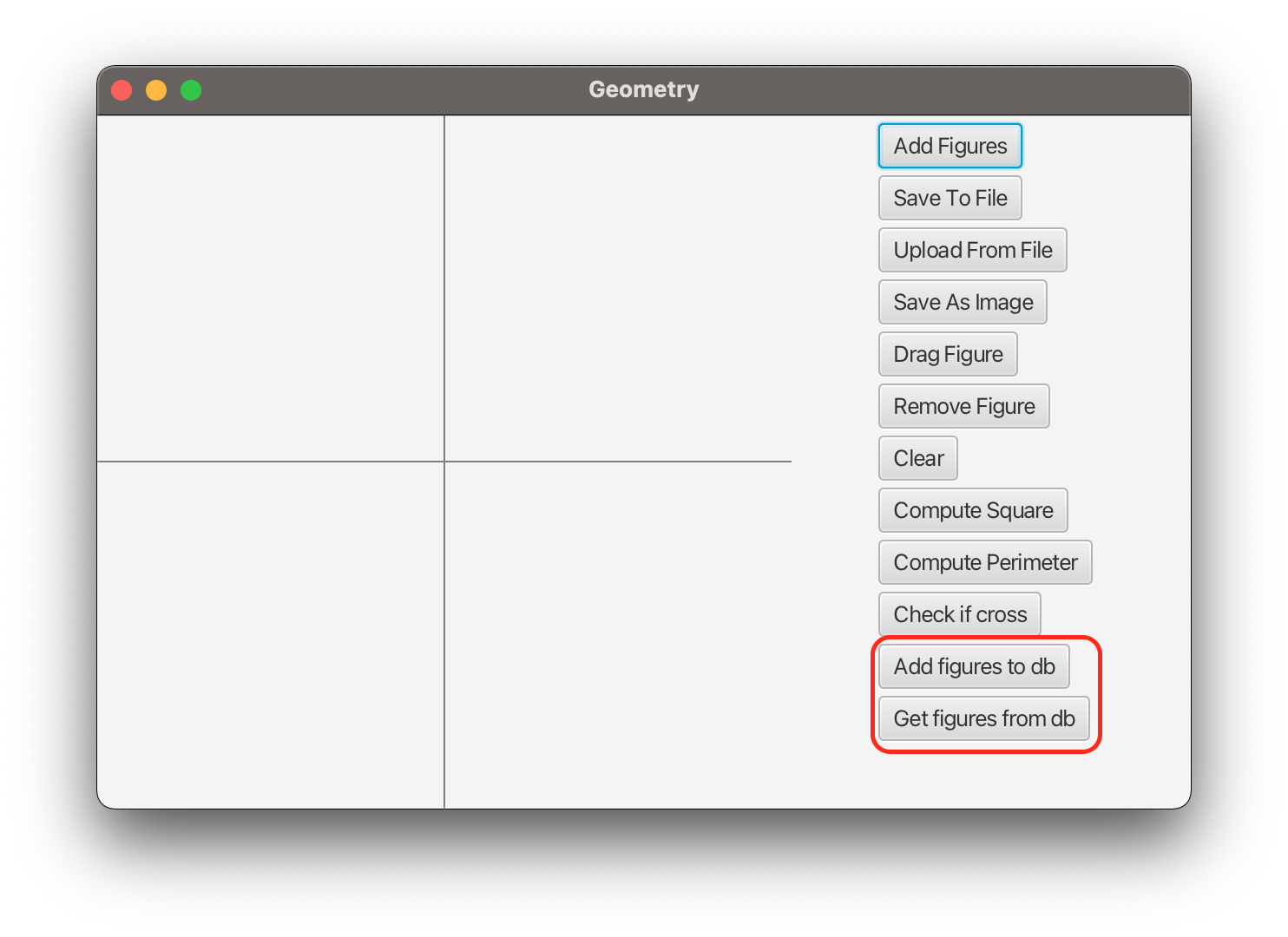
В рамках четвертой работы необходимо было внедрить Lombok и MongoDB. Аннотации @ToString не были добавлены, так как собственный метод был очень сложен.

Были введены аннотации @Getter, @Setter. Lombok позволяет избавиться от написания рутинных однотипных конструкций языка Java.

Также был добавлен метод bsonDocument, который возвращает тип данных документ, который используется для хранения в базе данных MongoDB.

Во второй проект было добавлено логирование, используя аннотацию @Log, всех процессов: добавление/удаление/перемещение фигур, вычисление над фигурами.

Также был добавлен функционал для сохранения всех фигур в MongoDB и загрузки из него, были добавлены кнопки для загрузки и выгрузки из базы данных. При старте программы происходит подключение к БД, получение клиента, с помощью которого можно работать с БД.



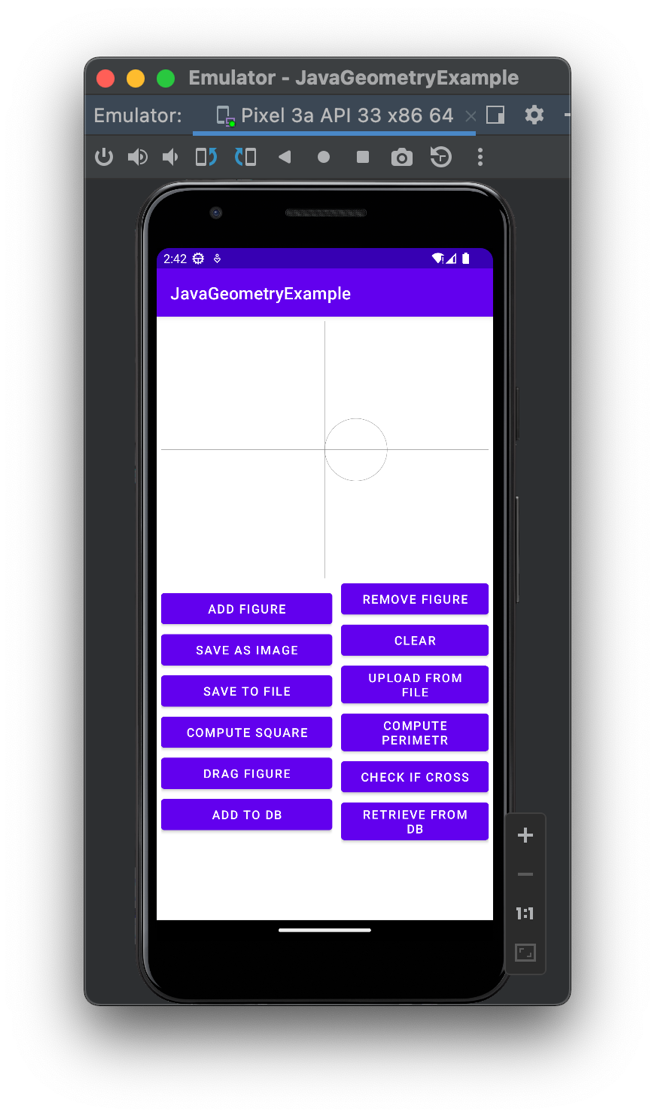
К базе данных можно было подключиться на сайте и смотреть за изменениями, которые могли происходить:

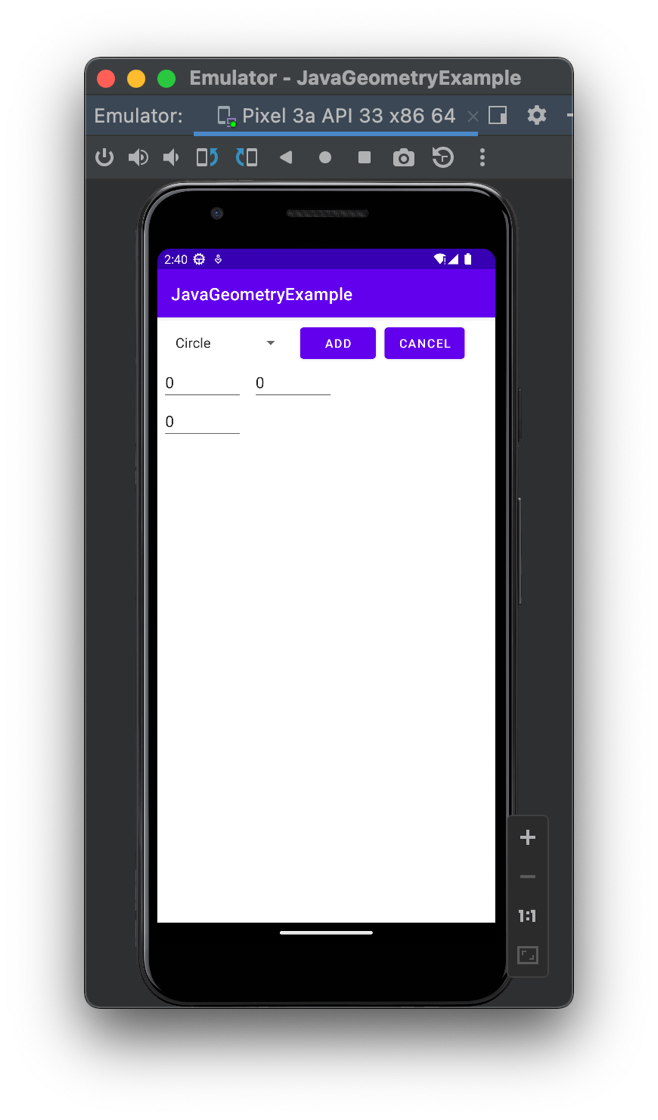
Изображение выглядит как текст

Автоматически созданное описание

# Семестровая (5 часть)

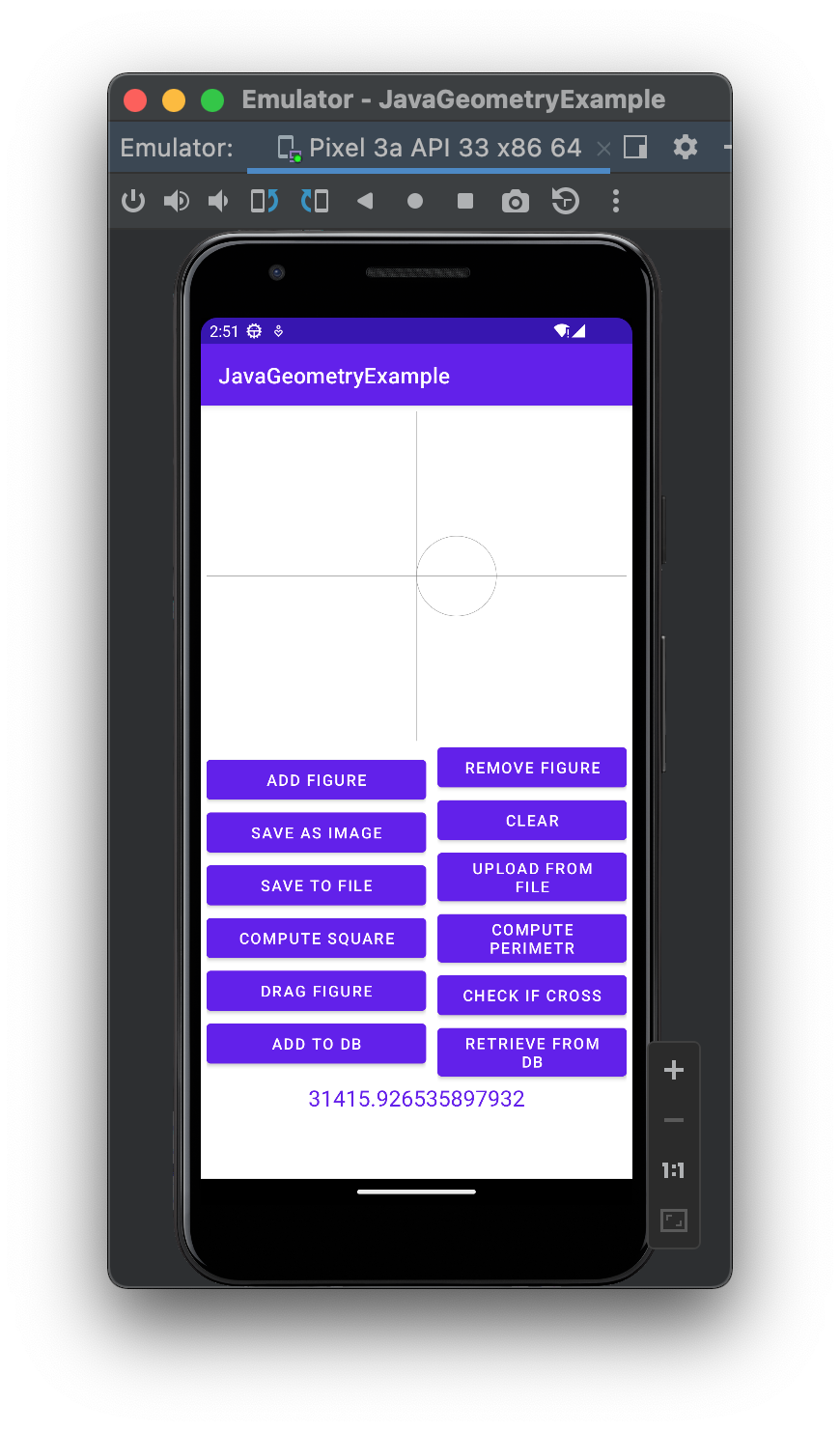
Требовалось реализовать android-приложение. В android вместо окон существуют activity. Работа с андроид специфична. Для передачи данных используется Activity Result API. Создаются своего рода контракты, с помощью которых можно отправлять данные в другую форму. Можно открывать диалоги выбора для сохранения, открытия файлов.



Добавление геометрической фигуры:

Картинка, построенная внутри программы, сохраняется в галерею приложения:

Вычисление площади:



# Код

## Консольное приложение

Circle.java

package app;  
  
import org.bson.Document;  
  
public class Circle implements IShape {  
 private Point2D p;  
 private double r;  
 private String type;  
  
 @Override  
 public String getType() { return type; }  
  
 public Circle(Point2D p, double r) throws Exception {  
 this.p = new Point2D(p.getX(0), p.getX(1));  
 this.r = r;  
  
 type = "Circle";  
 }  
  
 public Point2D getP() {  
 return p;  
 }  
  
 public void setP(Point2D p) throws Exception {  
 this.p = new Point2D(p.getX(0), p.getX(1));  
 }  
  
 public double getR() {  
 return r;  
 }  
  
 public void setR(double r) throws Exception {  
 this.r = r;  
 }  
  
 @Override  
 public double square() {  
 return Math.*PI* \* r \* r;  
 }  
  
 @Override  
 public double length() throws Exception {  
 return 2 \* Math.*PI* \* r;  
 }  
  
 @Override  
 public Circle shift(Point2D a) throws Exception {  
 p.add(a);  
 return this;  
 }  
  
 @Override  
 public Circle rot(double phi) throws Exception {  
 p.rot(phi);  
 return this;  
 }  
  
 @Override  
 public Circle symAxis(int i) throws Exception {  
 p.symAxis(i);  
 return this;  
 }  
  
 @Override  
 public boolean cross(IShape s) throws Exception {  
 if (s instanceof Circle) {  
 Point2D curP = ((Circle) s).getP();  
 double curR = ((Circle) s).getR();  
 if ((Point2D.*sub*(curP, p)).abs() <= curR + r) {  
 return true;  
 }  
 return false;  
 }  
 return s.cross(this);  
 }  
  
 @Override  
 public Document toBson() {  
 Document circle = new Document("type", type)  
 .append("data", toString());  
  
 return circle;  
 }  
   
 @Override  
 public String toString() {  
 return String.*format*("Circle(p=%s, r=%.2f)", p.toString(), r);  
 }  
}

IFigure.java

package app;  
  
public interface IFigure {  
 double square() throws Exception;  
 double volume() throws Exception;  
}

IMoveable.java

package app;  
  
public interface IMoveable {  
 IMoveable shift(Point2D a) throws Exception;  
 IMoveable rot(double phi) throws Exception;  
 IMoveable symAxis(int i) throws Exception;  
}

IShape.java

package app;  
  
import org.bson.Document;  
  
public interface IShape {  
 double square() throws Exception;  
 double length() throws Exception;  
 IShape shift(Point2D a) throws Exception;  
 IShape rot(double phi) throws Exception;  
 IShape symAxis(int i) throws Exception;  
 <T extends IShape> boolean cross(T s) throws Exception;  
 Document toBson();  
 String getType();  
 double *centerX* = 200.0;  
 double *centerY* = 200.0;  
}

Main.java

package app;  
  
import java.io.BufferedReader;  
import java.io.IOException;  
import java.util.ArrayList;  
import java.io.InputStreamReader;  
import java.util.HashSet;  
  
public class Main {  
  
 // список фигур  
 static ArrayList<IShape> *figures* = new ArrayList<IShape>();  
 // список дополнительных фигур для проверки на пересечение  
 static ArrayList<IShape> *crossFigures* = new ArrayList<IShape>();  
 // список названий фигур  
 static ArrayList<String> *figureNames* = new ArrayList<String>();  
 // для чтения инпута из консоли  
 static BufferedReader *reader* = new BufferedReader(new InputStreamReader(System.*in*));  
 // количество фигур  
 static int *n*;  
 // список возможных фигур  
 static HashSet<String> *figureTypes* = new HashSet<String>() {{  
 add("circle");  
 add("ngon");  
 add("polyline");  
 add("qgon");  
 add("rectangle");  
 add("segment");  
 add("tgon");  
 add("trapeze");  
 }};  
 // список возможных движений  
 static HashSet<String> *figureMovements* = new HashSet<String>() {{  
 add("rotate");  
 add("shift");  
 add("symmetry");  
 }};  
  
 // короткий вывод  
 static void print(String str) {  
 System.*out*.print(str);  
 }  
  
 // короткий вывод для фигуры  
 static void print(IShape figure) {  
 System.*out*.println(figure.toString());  
 }  
  
 // короткое чтение строки  
 static String read() throws IOException {  
 return *reader*.readLine();  
 }  
  
 // выходим из программы, если критическая ошибка при чтении  
 static void returnOnIOEXception() {  
 *print*("Something went wrong");  
 System.*exit*(1);  
 }  
  
 // обработать движение и применить к фигуре  
 static IShape readMovementAndApplyToFigure(String movementType, IShape figure) throws Exception {  
 *print*("\n");  
 switch (movementType) {  
 case "rotate":  
 return *applyRotateToFigure*(figure);  
 case "symmetry":  
 return *applySymmetryToFigure*(figure);  
 case "shift":  
 return *applyShiftToFigure*(figure);  
 }  
  
 throw new Exception("Bad data");  
 }  
  
 static IShape applyRotateToFigure(IShape figure) throws Exception {  
 *print*("Enter rotation angle (in radians):\n");  
  
 double phi = Double.*parseDouble*(*read*());  
 return figure.rot(phi);  
 }  
  
 static IShape applySymmetryToFigure(IShape figure) throws Exception {  
 *print*("Enter number of axis of symmetry:\n");  
  
 int axis = Integer.*parseInt*(*read*());  
 return figure.symAxis(axis);  
 }  
  
 static IShape applyShiftToFigure(IShape figure) throws Exception {  
 *print*("Enter vector:\n");  
  
 Point2D point = *readPoint*();  
 return figure.shift(point);  
 }  
  
 // обработать фигуру и ввести ее  
 static IShape readFigure(String figureType) throws Exception {  
 *print*("\n");  
 switch (figureType) {  
 case "circle":  
 return *readCircle*();  
 case "ngon":  
 return *readNGon*();  
 case "polyline":  
 return *readPolyline*();  
 case "qgon":  
 return *readQGon*();  
 case "rectangle":  
 return *readRectangle*();  
 case "segment":  
 return *readSegment*();  
 case "tgon":  
 return *readTGon*();   
 case "trapeze":  
 return *readTrapeze*();  
 }  
   
 throw new IOException("There is no such type");  
 }  
  
 // короткая запись ввода точки  
 static Point2D readPoint() throws Exception {  
 String[] point = *read*().split(" ");  
 if (point.length != 2) {  
 throw new IOException("Wrong input");  
 }  
  
 return new Point2D(Double.*parseDouble*(point[0]), Double.*parseDouble*(point[1]));  
 }  
  
 // короткое чтение нескольких точек  
 static Point2D[] readManyPoints(int n) throws Exception {  
 *print*(String.*format*("Enter %d points:\n", n));  
 Point2D[] points = new Point2D[n];  
 for (int i = 0; i < n; i++) {  
 points[i] = *readPoint*();  
 }  
  
 return points;  
 }  
  
 static Circle readCircle() throws Exception {  
 *print*("Enter center of circle:\n");  
   
 Point2D center = *readPoint*();  
  
 *print*("Enter radius:\n");  
 double r = Double.*parseDouble*(*read*());  
  
 return new Circle(center, r);  
 }  
  
 static NGon readNGon() throws Exception {  
 *print*("Enter count of points:\n");  
 int n = Integer.*parseInt*(*read*());  
  
 return new NGon(*readManyPoints*(n));  
 }  
  
 static Polyline readPolyline() throws Exception {  
 *print*("Enter count of points:\n");  
 int n = Integer.*parseInt*(*read*());  
  
 return new Polyline(*readManyPoints*(n));  
 }  
  
 static QGon readQGon() throws Exception {  
 int n = 4;  
 return new QGon(*readManyPoints*(n));  
 }  
  
 static Rectangle readRectangle() throws Exception {  
 int n = 4;  
 return new Rectangle(*readManyPoints*(n));  
 }  
  
 static Segment readSegment() throws Exception {  
 int n = 2;  
 Point2D[] points = *readManyPoints*(n);  
 return new Segment(points[0], points[1]);  
 }  
  
 static TGon readTGon() throws Exception {  
 int n = 3;  
 return new TGon(*readManyPoints*(n));  
 }  
  
 static Trapeze readTrapeze() throws Exception {  
 int n = 4;  
 return new Trapeze(*readManyPoints*(n));  
 }  
  
 // вводим фигуры и обрабатываем запрос (3 и 4 пункты)  
 static void readFigures(int n) {  
 for (int i = 0; i < n; i++) {  
 *print*("Enter type of possible figure:\n");  
 String figureType = "";  
 try {  
 figureType = *read*().toLowerCase();  
 } catch (IOException e) {  
 *print*(e.getMessage() + "\n");  
 i--;  
 continue;  
 }  
  
 if (!*figureTypes*.contains(figureType)) {  
 *print*("Input is wrong. Try again:\n");  
 i--;  
 continue;  
 }  
  
 IShape figure;  
 try {  
 figure = *readFigure*(figureType);  
 } catch (Exception e){  
 *print*(e.getMessage() + "\n");  
 i--;  
 continue;  
 }  
  
 *figures*.add(figure);  
 *figureNames*.add(figureType);  
 }  
 }  
  
 // 5 пункт  
 static void checkNewFigures(int n) {  
 for (int i = 0; i < n; i++) {  
 try {  
 IShape figure = *readFigure*(*figureNames*.get(i));  
 if (figure.cross(*figures*.get(i))) {  
 *print*("Cross\n");  
 } else {  
 *print*("No Cross\n");  
 }  
  
 *print*("Enter possible movement:\n");  
 String movementType = "";  
 try {  
 movementType = *read*().toLowerCase();  
 } catch (IOException e) {  
 *print*(e.getMessage() + "\n");  
 i--;  
 continue;  
 }  
  
 if (!*figureMovements*.contains(movementType)) {  
 *print*("Input is wrong:\n");  
 i--;  
 continue;  
 }  
  
 *readMovementAndApplyToFigure*(movementType, figure);  
 if (figure.cross(*figures*.get(i))) {  
 *print*("Cross\n");  
 } else {  
 *print*("No Cross\n");  
 }  
  
 } catch (Exception e){  
 *print*(e.getMessage() + "\n");  
 i--;  
 continue;  
 }  
 }  
 }  
  
 public static void main(String[] args) {  
 *print*("Enter number of figure:\n");  
   
 try {  
 *n* = Integer.*parseInt*(*read*());  
 } catch (IOException e) {  
 *returnOnIOEXception*();  
 }  
  
 *print*("You can input one figure of following:\n");  
 for (String figureType: *figureTypes*) {  
 *print*(figureType + "\n");  
 }  
  
 *readFigures*(*n*);  
  
 double square = 0.;  
 double length = 0.;  
 for (IShape figure: *figures*) {  
 try {  
 square += figure.square();  
 } catch (Exception e) {  
 *print*(e.getMessage() + "\n");  
 continue;  
 }  
  
 try {  
 length += figure.length();  
 } catch (Exception e) {  
 *print*(e.getMessage() + "\n");  
 continue;  
 }  
 }  
  
 *print*(String.*format*("Square: %.2f", square));  
 *print*(String.*format*("Length: %.2f", length));  
 *print*(String.*format*("Average square: %.2f", square / *n*));  
  
 *checkNewFigures*(*n*);  
 }  
}

NGon.java

package app;  
  
import org.bson.Document;  
  
public class NGon implements IShape {  
 protected int n;  
 protected Point2D[] p;  
 protected String type;  
  
 @Override  
 public String getType() {  
 return type;  
 }  
  
 public NGon(Point2D[] p) throws Exception {  
 if (p.length <= 2)  
 throw new Exception("NGon: Array of points is too small\n");  
 this.n = p.length;  
 this.p = new Point2D[p.length];  
 for (int i = 0; i < n; i++) {  
 this.p[i] = new Point2D(p[i].getX(0), p[i].getX(1));  
 }  
 type = "NGon";  
 }  
  
 public int getN() {  
 return n;  
 }  
  
 public Point2D[] getP() {  
 return p;  
 }  
  
 public Point2D getP(int pos) throws Exception {  
 if (pos < 0 || pos >= n)  
 throw new Exception("NGon: Pos in getP is not valid\n");  
  
 return p[pos];  
 }  
  
 public void setP(Point2D[] p) throws Exception {  
 if (p.length != n)  
 throw new Exception("NGon: Array size in setP is not correct\n");  
  
 this.p = p;  
 }  
  
 public void setP(Point2D p, int pos) throws Exception {  
 if (pos < 0 || pos >= n)  
 throw new Exception("NGon: Pos in setP is not correct\n");  
  
 this.p[pos] = p;  
 }  
  
 @Override  
 public double square() throws Exception {  
 double s = 0.;  
 for (int i = 2; i < n; i++) {  
 double a = Point2D.*sub*(p[0], p[i - 1]).abs();  
 double b = Point2D.*sub*(p[0], p[i]).abs();  
 double c = Point2D.*sub*(p[i], p[i - 1]).abs();  
 double halfP = (a + b + c) / 2;  
 s += Math.*sqrt*(halfP \* (halfP - a) \* (halfP - b) \* (halfP - c));  
 }  
  
 return s;  
 }  
  
 @Override  
 public double length() throws Exception {  
 double len = 0;  
 for (int i = 1; i < n; i++) {  
 len += Point2D.*sub*(p[i - 1], p[i]).abs();  
 }  
 len += Point2D.*sub*(p[n - 1], p[0]).abs();  
 return len;  
 }  
  
 @Override  
 public NGon shift(Point2D p) throws Exception {  
 Point2D pp = new Point2D(p.getX(0), p.getX(1));  
 for (int i = 0; i < n; i++) {  
 this.p[i].add(pp);  
 }  
  
 return this;  
 }  
  
 @Override  
 public NGon rot(double phi) throws Exception {  
 for (int i = 0; i < n; i++) {  
 this.p[i].rot(phi);  
 }  
  
 return this;  
 }  
  
 @Override  
 public NGon symAxis(int i) throws Exception {  
 for (int j = 0; j < n; j++) {  
 this.p[j].symAxis(i);  
 }  
  
 return this;  
 }  
  
  
 @Override  
 public boolean cross(IShape shape) throws Exception {  
 Segment seg;  
 for (int i = 1; i < n; i++) {  
 seg = new Segment(p[i - 1], p[i]);  
 if (seg.cross(shape)) {  
 return true;  
 }  
 }  
  
 seg = new Segment(p[n - 1], p[0]);  
 if (seg.cross(shape)) {  
 return true;  
 }  
  
 return false;  
 }  
  
 @Override  
 public Document toBson() {  
 Document ngon = new Document("type", type)  
 .append("data", toString());  
  
 return ngon;  
 }  
  
 @Override  
 public String toString() {  
 StringBuilder str = new StringBuilder(String.*format*("NGon(n=%d, p=(", n));  
 String comma = ", ";  
 for (int i = 0; i < n; i++) {  
 if (i == n - 1)  
 comma = "";  
 str.append(p[i].toString() + comma);  
 }  
 str.append("))");  
 return str.toString();  
 }  
}

OpenFigure.java

package app;  
  
public abstract class OpenFigure implements IShape {  
 @Override  
 public double square() {  
 return 0;  
 }  
}

Point.java

package app;  
  
import org.bson.Document;  
  
public class Point {  
 protected int dim;  
 protected double[] x;  
 protected String type = "Point";  
  
 public int getDim() {  
 return dim;  
 }  
  
 public double[] getX() {  
 return x;  
 }  
  
 public double getX(int pos) throws Exception {  
 if (pos >= dim || pos < 0) {  
 throw new Exception("Point: Position is not valid\n");  
 }  
  
 return x[pos];  
 }  
  
 public void setX(double[] newX) throws Exception {  
 if (newX.length != dim) {  
 throw new Exception("Point: New coordinates have incorrect dimension\n");  
 }  
  
 x = newX;  
 }  
  
 public void setX(int pos, double newX) throws Exception {  
 if (pos < 0 || pos >= dim) {  
 throw new Exception("Point: Position to update is not valid\n");  
 }  
  
 x[pos] = newX;  
 }  
  
 public Point(int dim) throws Exception {  
 if (dim < 0) {  
 throw new Exception("Point: Dimension cannot be negative\n");  
 }  
  
 this.dim = dim;  
 this.x = new double[dim];  
 for (int i = 0; i < dim; i++)  
 x[i] = 0.0;  
 }  
  
 public Point(int dim, double[] arr) throws Exception {  
 if (dim < 0) {  
 throw new Exception("Point: Dimension cannot be negative\n");  
 }  
  
 if (arr.length != dim) {  
 throw new Exception("Point: Number of coordinates doesn't equal to dimension\n");  
 }  
  
 this.dim = dim;  
 this.x = arr;  
 }  
  
 public double abs() {  
 double rast = 0;  
 for (double c : x) {  
 rast += c \* c;  
 }  
  
 return Math.*sqrt*(rast);  
 }  
  
 public static Point add(Point a, Point b) throws Exception {  
 if (a.getDim() != b.getDim()) {  
 throw new Exception("Point: Cannot add two points\n");  
 }  
  
 Point res = new Point(a.getDim());  
 for (int i = 0; i < res.getDim(); i++) {  
 res.setX(i, a.getX(i) + b.getX(i));  
 }  
  
 return res;  
 }  
  
 public void add(Point b) throws Exception {  
 if (dim != b.getDim()) {  
 throw new Exception("Point: Cannot add a point\n");  
 }  
  
 for (int i = 0; i < dim; i++) {  
 x[i] += b.getX(i);  
 }  
 }  
  
 public static Point sub(Point a, Point b) throws Exception {  
 if (a.getDim() != b.getDim()) {  
 throw new Exception("Point: Cannot sub two points\n");  
 }  
  
 Point res = new Point(a.getDim());  
 for (int i = 0; i < res.getDim(); i++) {  
 res.setX(i, a.getX(i) - b.getX(i));  
 }  
  
 return res;  
 }  
  
 public void sub(Point b) throws Exception {  
 if (dim != b.getDim()) {  
 throw new Exception("Point: Cannot sub a point\n");  
 }  
  
 for (int i = 0; i < dim; i++) {  
 x[i] -= b.getX(i);  
 }  
 }  
  
 public static double mult(Point a, Point b) throws Exception {  
 if (a.getDim() != b.getDim()) {  
 throw new Exception("Point: Cannot multi two points\n");  
 }  
  
 double res = 0.0;  
 for (int i = 0; i < a.getDim(); i++) {  
 res += a.getX(i) \* b.getX(i);  
 }  
  
 return res;  
 }  
  
 public static Point mult(Point a, double r) throws Exception {  
 Point res = new Point(a.getDim());  
 for (int i = 0; i < a.getDim(); i++) {  
 res.setX(i, a.getX(i) \* r);  
 }  
  
 return res;  
 }  
  
 public double multi(Point a) throws Exception {  
 if (a.getDim() != dim) {  
 throw new Exception("Point: Cannot multi to a point\n");  
 }  
  
 double res = 0.0;  
 for (int i = 0; i < dim; i++) {  
 res += a.getX(i) \* x[i];  
 }  
  
 return res;  
 }  
  
 public Point multi(double r) {  
 for (int i = 0; i < dim; i++) {  
 x[i] \*= r;  
 }  
  
 return this;  
 }  
  
 public static Point symAxis(Point a, int pos) throws Exception {  
 if (a.getDim() <= pos || pos < 0) {  
 throw new Exception("Point: Cannot symAxis point\n");  
 }  
  
 Point res = new Point(a.getDim());  
 for (int i = 0; i < a.getDim(); i++) {  
 if (i == pos) {  
 res.setX(i, a.getX(i));  
 continue;  
 }  
 res.setX(i, -a.getX(i));  
 }  
  
 return res;  
 }  
  
 public Point symAxis(int pos) throws Exception {  
 if (pos < 0 || pos >= dim) {  
 throw new Exception("Point: Cannot symAxis to a point");  
 }  
  
 for (int i = 0; i < dim; i++) {  
 if (i == pos) {  
 continue;  
 }  
 x[i] \*= (-1);  
 }  
  
 return this;  
 }  
  
 public Document toBson() {  
 return new Document("type", type)  
 .append("data", toString());  
 }  
  
 @Override  
 public String toString() {  
 StringBuilder str = new StringBuilder(String.*format*("Point(dims=%d, x=(", dim));  
 String comma = ", ";  
 for (int i = 0; i < x.length; i++) {  
 if (i == x.length - 1)  
 comma = "";  
 str.append(x[i]).append(comma);  
 }  
  
 str.append("))");  
  
 return str.toString();  
 }  
  
}

Point2D.java

package app;  
  
import java.lang.Math;  
  
public class Point2D extends Point {  
 public Point2D() throws Exception {  
 super(2);  
 }  
  
 public Point2D(double[] x) throws Exception {  
 super(2, x);  
 }  
  
 public Point2D(double x, double y) throws Exception {  
 super(2, new double[] {x, y});  
 }  
  
 public static Point2D rot(Point2D p, double phi) throws Exception {  
 double angle;  
 if (p.abs() == 0)  
 return p;  
 if (p.getX(0) >= 0) {  
 angle = Math.*asin*(p.getX(1) / p.abs());  
 } else {  
 angle = Math.*PI* - Math.*asin*(p.getX(1) / p.abs());  
 }  
  
 Point2D pp = new Point2D(p.abs() \* Math.*cos*(angle + phi), p.abs() \* Math.*sin*(angle + phi));  
 return pp;  
 }  
  
 public Point2D rot(double phi) throws Exception {  
 double angle;  
 if (abs() == 0)  
 return this;  
 if (x[0] >= 0) {  
 angle = Math.*asin*(x[1] / abs());  
 } else {  
 angle = Math.*PI* - Math.*asin*(x[1] / abs());  
 }  
  
 double xx = abs() \* Math.*cos*(angle + phi);  
 double yy = abs() \* Math.*sin*(angle + phi);  
  
 x[0] = xx;  
 x[1] = yy;  
  
 return this;  
 }  
}

Point3D.java

package app;  
  
public class Point3D extends Point {  
 public Point3D() throws Exception {  
 super(3);  
 }  
  
 public Point3D(double[] x) throws Exception {  
 super(3, x);  
 }  
  
 public static Point3D cross\_prod(Point3D p1, Point3D p2) throws Exception {  
 Point3D res = new Point3D();  
 res.setX(0, p1.getX(1) \* p2.getX(2) - p1.getX(2) \* p2.getX(1));  
 res.setX(1, p1.getX(0) \* p2.getX(2) - p1.getX(2) \* p2.getX(0));  
 res.setX(2, p1.getX(0) \* p2.getX(1) - p1.getX(1) \* p2.getX(0));  
 return res;  
 }  
  
 public Point3D cross\_prod(Point3D p2) throws Exception {  
 this.setX(0, this.getX(1) \* p2.getX(2) - this.getX(2) \* p2.getX(1));  
 this.setX(1, this.getX(0) \* p2.getX(2) - this.getX(2) \* p2.getX(0));  
 this.setX(2, this.getX(0) \* p2.getX(1) - this.getX(1) \* p2.getX(0));  
 return new Point3D(x);  
 }  
  
 public static double mix\_prod(Point3D p1, Point3D p2, Point3D p3) throws Exception {  
 return p1.getX(0) \* p2.getX(1) \* p3.getX(2)  
 + p1.getX(1) \* p2.getX(2) \* p3.getX(0)  
 + p1.getX(2) \* p2.getX(0) \* p3.getX(1)  
 - p1.getX(2) \* p2.getX(1) \* p3.getX(0)  
 - p1.getX(0) \* p2.getX(2) \* p3.getX(1)  
 - p1.getX(1) \* p2.getX(0) \* p3.getX(2);  
 }  
  
 public double mix\_prod(Point3D p2, Point3D p3) throws Exception {  
 return x[0] \* p2.getX(1) \* p3.getX(2)  
 + x[1] \* p2.getX(2) \* p3.getX(0)  
 + x[2] \* p2.getX(0) \* p3.getX(1)  
 - x[2] \* p2.getX(1) \* p3.getX(0)  
 - x[0] \* p2.getX(2) \* p3.getX(1)  
 - x[1] \* p2.getX(0) \* p3.getX(2);  
 }  
}

Polyline.java

package app;  
  
import org.bson.Document;  
  
public class Polyline extends OpenFigure {  
 private int n;  
 private Point2D[] p;  
 private String type;  
  
 @Override  
 public String getType() { return type; }  
  
 public Polyline(Point2D[] p) throws Exception {  
 this.p = new Point2D[p.length];  
 for (int i = 0; i < p.length; i++) {  
 this.p[i] = new Point2D(p[i].getX(0), p[i].getX(1));  
 }  
 this.n = p.length;  
 type = "Polyline";  
 }  
  
 public int getN() {  
 return n;  
 }  
  
 public Point2D[] getP() {  
 return p;  
 }  
  
 public Point2D getP(int pos) throws Exception {  
 if (pos < 0 || pos >= n)  
 throw new Exception("Polyline: Position is not valid\n");  
  
 return p[pos];  
 }  
  
 public void setP(Point2D[] arr) throws Exception {  
 if (n != arr.length)  
 throw new Exception("Polyline: Array is not valid\n");  
  
 p = arr;  
 }  
  
 public void setP(Point2D p, int pos) throws Exception {  
 if (pos < 0 || pos >= n)  
 throw new Exception("Polyline: Position to set is not valid\n");  
  
 this.p[pos] = p;  
 }  
  
 @Override  
 public double length() throws Exception {  
 double res = 0;  
 for (int i = 1; i < n; i++) {  
 res += (new Segment(p[i - 1], p[i])).length();  
 }  
 return res;  
 }  
  
 @Override  
 public Polyline shift(Point2D a) throws Exception {  
 for (int i = 0; i < n; i++) {  
 p[i].add(a);  
 }  
  
 return this;  
 }  
  
 @Override  
 public Polyline rot(double phi) throws Exception {  
 for (int i = 0; i < n; i++) {  
 p[i].rot(phi);  
 }  
  
 return this;  
 }  
  
 @Override  
 public Polyline symAxis(int i) throws Exception {  
 for (int j = 0; j < n; j++) {  
 p[j].symAxis(i);  
 }  
  
 return this;  
 }  
  
 @Override  
 public boolean cross(IShape shape) throws Exception {  
 for (int i = 1; i < n; i++) {  
 Segment seg = new Segment(p[i - 1], p[i]);  
 if (seg.cross(shape))  
 return true;  
 }  
  
 return false;  
 }  
  
 @Override  
 public Document toBson() {  
 return new Document("type", type)  
 .append("data", toString());  
 }  
  
 @Override  
 public String toString() {  
 StringBuilder str = new StringBuilder(String.*format*("Polyline(n=%d, p=(", n));  
 String comma = ", ";  
 for (int i = 0; i < n; i++) {  
 if (i == n - 1)  
 comma = "";  
 str.append(p[i].toString()).append(comma);  
 }  
 str.append("))");  
 return str.toString();  
 }  
}

QGon.java

package app;  
  
import org.bson.Document;  
  
public class QGon extends NGon {  
 public QGon(Point2D[] p) throws Exception {  
 super(p);  
 type = "QGon";  
 if (n != 4)  
 throw new Exception("QGon: Array size is not correct\n");  
 }  
  
 @Override  
 public double square() throws Exception {  
 return super.square();  
 }  
  
 @Override  
 public Document toBson() {  
 return new Document("type", type)  
 .append("data", toString());  
 }  
  
 @Override  
 public String toString() {  
 return type + super.toString().substring(4);  
 }  
}

Rectangle.java

package app;  
  
import org.bson.Document;  
  
public class Rectangle extends QGon {  
 public Rectangle(Point2D[] p) throws Exception {  
 super(p);  
 type = "Rectangle";  
 if (Point2D.*mult*(Point2D.*sub*(p[n - 1], p[0]), Point2D.*sub*(p[0], p[1])) != 0) {  
 throw new Exception("Rectangle: Point is not valid\n");  
 }  
 for (int i = 1; i < n - 1; i++) {  
 if (Point2D.*mult*(Point2D.*sub*(p[i - 1], p[i]), Point2D.*sub*(p[i], p[i + 1])) != 0) {  
 throw new Exception("Rectangle: Point is not valid\n");  
 }  
 }  
 if (Point2D.*mult*(Point2D.*sub*(p[n - 2], p[n - 1]), Point2D.*sub*(p[n - 1], p[0])) != 0) {  
 throw new Exception("Rectangle: Point is not valid\n");  
 }  
 }  
  
 @Override  
 public double square() throws Exception {  
 return Point2D.*sub*(p[0], p[1]).abs() \* Point2D.*sub*(p[1], p[2]).abs();  
 }  
  
 @Override  
 public Document toBson() {  
 Document rectangle = new Document("type", type)  
 .append("data", toString());  
  
 return rectangle;  
 }  
  
 @Override  
 public String toString() {  
 return super.toString();  
 }  
}

Segment.java

package app;  
  
import org.bson.Document;  
  
public class Segment extends OpenFigure {  
 private Point2D start;  
 private Point2D finish;  
 private String type;  
  
 @Override  
 public String getType() { return type; }  
  
 public Segment(Point2D a, Point2D b) throws Exception {  
 if (a.getX(0) == b.getX(0) && a.getX(1) == b.getX(1)) {  
 throw new Exception("Segment: Start and finish are the same point\n");  
 }  
 start = new Point2D(a.getX(0), a.getX(1));  
 finish = new Point2D(b.getX(0), b.getX(1));  
 type = "Segment";  
 }  
  
 public Point2D getStart() {  
 return start;  
 }  
  
 public Point2D getFinish() {  
 return finish;  
 }  
  
 public void setStart(Point2D p) throws Exception {  
 this.start = new Point2D(p.getX(0), p.getX(1));  
 }  
  
 public void setFinish(Point2D p) throws Exception {  
 this.finish = new Point2D(p.getX(0), p.getX(1));  
 }  
  
 @Override  
 public double length() throws Exception {  
 return (new Point2D(Point.*sub*(start, finish).getX())).abs();  
 }  
  
 @Override  
 public Segment shift(Point2D a) throws Exception {  
 a.setX(1, a.getX(1));  
 start.add(a);  
 finish.add(a);  
 return this;  
 }  
  
 @Override  
 public Segment rot(double phi) throws Exception {  
 start.rot(phi);  
 finish.rot(phi);  
 return this;  
 }  
  
 @Override  
 public Segment symAxis(int i) throws Exception {  
 start.symAxis(i);  
 finish.symAxis(i);  
 return this;  
 }  
  
 @Override  
 public boolean cross(IShape i) throws Exception {  
 if (i instanceof Segment) {  
 Segment seg = (Segment)i;  
 Point2D a = new Point2D(start.getX(0), start.getX(1));  
 Point2D b = new Point2D(finish.getX(0), finish.getX(1));  
 Point2D c = new Point2D(seg.getStart().getX(0), seg.getStart().getX(1));  
 Point2D d = new Point2D(seg.getFinish().getX(0), seg.getFinish().getX(1));  
  
 Point2D p1 = new Point2D(Point.*sub*(b, a).getX(0), Point.*sub*(b, a).getX(1));  
 Point2D p2 = new Point2D(Point.*sub*(d, c).getX(0), Point.*sub*(d, c).getX(1));  
  
 double x1 = a.getX(0), x2 = b.getX(0), x3 = c.getX(0), x4 = d.getX(0);  
 double y1 = a.getX(1), y2 = b.getX(1), y3 = c.getX(1), y4 = d.getX(1);  
  
 if (Math.*abs*(p1.getX(0) \* p2.getX(0) + p1.getX(1) \* p2.getX(1) - p1.abs() \* p2.abs()) <= 0.001) {  
 Point2D additionalPoint = new Point2D(x3 - x1, y3 - y1);  
 if ((Math.*abs*(  
 p1.getX(0) \* additionalPoint.getX(0)  
 + p1.getX(1) \* additionalPoint.getX(1)  
 - p1.abs() \* additionalPoint.abs()  
 ) <= 0.001 ||  
 Math.*abs*(  
 p1.getX(0) \* additionalPoint.getX(0)  
 + p1.getX(1) \* additionalPoint.getX(1)  
 + p1.abs() \* additionalPoint.abs()  
 ) <= 0.001)  
 && (x1 <= x3 && x3 <= x2 || x1 <= x4 && x4 <= x2 || x2 <= x3 && x3 <= x1 || x2 <= x4 && x4 <= x1)) {  
 return true;  
 } else {  
 return false;  
 }  
 }  
  
 double k2Up = (x2 - x1) \* y1 + (x3 - x1) \* (y2 - y1) - y3 \* (x2 - x1);  
 double k2Down = (y4 - y3) \* (x2 - x1) - (y2 - y1) \* (x4 - x3);  
 double k2 = k2Up / k2Down;  
  
 Point result = p2.multi(k2);  
 result.add(c);  
  
 if (  
 (x3 <= result.getX(0) && result.getX(0) <= x4 || x4 <= result.getX(0) && result.getX(0) <= x3)  
 &&  
 (x1 <= result.getX(0) && result.getX(0) <= x2 || x2 <= result.getX(0) && result.getX(0) <= x1)  
 ) {  
 return true;  
 }  
  
 return false;  
 } else if (i instanceof Polyline) {  
 Point2D[] coords = ((Polyline) i).getP();  
 Segment seg;  
 for (int j = 1; j < coords.length; j++) {  
 seg = new Segment(coords[j - 1], coords[j]);  
 if (seg.cross(this))  
 return true;  
 }  
 seg = new Segment(coords[coords.length - 1], coords[0]);  
 return seg.cross(this);  
 } else if (i instanceof NGon) {  
 Point2D[] coords = ((NGon) i).getP();  
 Segment seg;  
  
 double[] arr1 = new double[coords.length];  
 double[] arr2 = new double[coords.length];  
  
 for (int j = 1; j < coords.length; j++) {  
 seg = new Segment(coords[j - 1], coords[j]);  
 arr1[j - 1] = (start.getX(0) - coords[j - 1].getX(0)) \* (coords[j].getX(1) - coords[j - 1].getX(1))  
 - (start.getX(1) - coords[j - 1].getX(1)) \* (coords[j].getX(0) - coords[j - 1].getX(0));  
 arr2[j - 1] = (finish.getX(0) - coords[j - 1].getX(0)) \* (coords[j].getX(1) - coords[j - 1].getX(1))  
 - (finish.getX(1) - coords[j - 1].getX(1)) \* (coords[j].getX(0) - coords[j - 1].getX(0));  
 if (seg.cross(this))  
 return true;  
 }  
 seg = new Segment(coords[coords.length - 1], coords[0]);  
 arr1[coords.length - 1] = (start.getX(0) - coords[coords.length - 1].getX(0)) \* (coords[0].getX(1) - coords[coords.length - 1].getX(1))  
 - (start.getX(1) - coords[coords.length - 1].getX(1)) \* (coords[0].getX(0) - coords[coords.length - 1].getX(0));  
 arr2[coords.length - 1] = (finish.getX(0) - coords[coords.length - 1].getX(0)) \* (coords[0].getX(1) - coords[coords.length - 1].getX(1))  
 - (finish.getX(1) - coords[coords.length - 1].getX(1)) \* (coords[0].getX(0) - coords[coords.length - 1].getX(0));  
  
 boolean checker = true;  
  
 for (int k = 1; k < coords.length; k++) {  
 if (arr1[0] \* arr1[k] < 0) {  
 checker = false;  
 break;  
  
  
 }  
  
 if (arr2[0] \* arr2[k] < 0) {  
 checker = false;  
 break;  
 }  
 }  
  
 return (checker | seg.cross(this));  
 } else {  
 // this is for Circle  
  
 Point2D c = ((Circle) i).getP();  
 double r = ((Circle) i).getR();  
  
 if ((new Segment(c, start)).length() <= r || (new Segment(c, finish)).length() <= r) {  
 return true;  
 }  
 return false;  
 }  
 }  
  
 @Override  
 public Document toBson() {  
 Document segment = new Document("type", type)  
 .append("data", toString());  
  
 return segment;  
 }  
  
 @Override  
 public String toString() {  
 return String.*format*("Segment(start=%s, finish=%s)", start.toString(), finish.toString());  
 }  
}

TGon.java

package app;  
  
import org.bson.Document;  
  
public class TGon extends NGon {  
 public TGon(Point2D[] p) throws Exception {  
 super(p);  
 type = "TGon";  
 if (p.length != 3)  
 throw new Exception("TGon: Array size is not correct\n");  
 }  
  
 @Override  
 public double square() throws Exception {  
 double s = 0.0;  
 double a = Point2D.*sub*(p[0], p[1]).abs();  
 double b = Point2D.*sub*(p[0], p[2]).abs();  
 double c = Point2D.*sub*(p[2], p[1]).abs();  
 double halfP = (a + b + c) / 2;  
 s += Math.*sqrt*(halfP \* (halfP - a) \* (halfP - b) \* (halfP - c));  
 return s;  
 }  
  
 @Override  
 public Document toBson() {  
 Document tgon = new Document("type", type)  
 .append("data", toString());  
  
 return tgon;  
 }  
  
 @Override  
 public String toString() {  
 return type + super.toString().substring(4);  
 }  
}

Trapeze.java

package app;  
  
import org.bson.Document;  
  
public class Trapeze extends QGon {  
 public Trapeze(Point2D[] p) throws Exception {  
 super(p);  
 type = "Trapeze";  
 }  
  
 @Override  
 public double square() throws Exception {  
 return super.square();  
 }  
  
 @Override  
 public Document toBson() {  
 Document trapeze = new Document("type", type)  
 .append("data", toString());  
  
 return trapeze;  
 }  
  
 @Override  
 public String toString() {  
 return type + super.toString().substring(4);  
 }  
}

## Графическое приложение

В графическом приложении был использован модуль из консольного приложения, поэтому я не буду его дублировать снова.

Корневая папка:

build.gradle:

subprojects **{** apply plugin: 'application'  
 apply plugin: 'java'  
 repositories **{** jcenter()  
 mavenCentral()  
 **}** dependencies **{** implementation 'org.mongodb:mongodb-driver-sync:4.1.2'  
 testImplementation 'junit:junit:4.12'  
 testRuntimeOnly("org.junit.vintage:junit-vintage-engine:5.8.2")  
 **}** application **{** mainClass = 'ui.Main'  
 **}** tasks.named('test') **{** useJUnitPlatform()   
 **}  
  
}**tasks.register('runUi') **{** dependsOn ':data-module:build'  
 dependsOn ':data-module:test'  
 dependsOn ':presentation-module:build'  
 dependsOn ':presentation-module:run'  
**}**defaultTasks 'runUi'

rootProject.name = "java\_geometry\_example"  
  
include 'presentation-module'  
include 'data-module'

Папка presentation-module:

build.gradle:

plugins **{** id 'java'  
 id 'application'  
 id 'org.openjfx.javafxplugin' version '0.0.13'  
 id "io.franzbecker.gradle-lombok" version '5.0.0'  
**}**javafx **{** version = "18"  
 modules = [ 'javafx.controls', 'javafx.fxml', 'javafx.graphics', 'javafx.swing' ]  
**}**lombok **{** version = '1.18.4'  
 sha256 = ""  
**}**mainClassName = 'ui.Main'  
  
dependencies **{** implementation project(':data-module')  
**}**

Папка main в папке presentation-module:

Count.java:

package ui;  
  
import java.io.FileInputStream;  
import java.io.IOException;  
   
import javafx.fxml.FXMLLoader;  
import javafx.scene.Scene;  
import javafx.scene.layout.Pane;  
import javafx.stage.Stage;  
   
public class Count {  
  
 public Count() { }  
 private Stage stage;  
 private Pane root;  
 private MainController mController;  
 private String fxmlDocPath = "./src/main/java/ui/Count.fxml";  
  
 public Stage getStage() { return stage; }  
 public Pane getPane() { return root; }  
 public MainController getMainController() { return mController; }  
  
 public void start(MainController mController, boolean type) throws IOException {  
 this.mController = mController;  
 stage = new Stage();  
 FXMLLoader loader = new FXMLLoader();  
 FileInputStream fxmlStream = new FileInputStream(fxmlDocPath);  
 CountController controller = new CountController(this, type);  
 loader.setController(controller);  
 root = (Pane) loader.load(fxmlStream);  
  
 Scene scene = new Scene(root);  
 stage.setScene(scene);  
 stage.setTitle("Compute");  
 stage.show();  
 controller.setDataToCombo();  
 }  
  
 public void onDestroy() {  
 stage.close();  
 }  
}

Count.fxml:

<?xml version="1.0" encoding="UTF-8"?>  
  
<?import javafx.collections.\*?>   
<?import javafx.scene.canvas.Canvas?>  
<?import javafx.scene.control.\*?>  
<?import javafx.scene.layout.\*?>  
<?import javafx.scene.image.ImageView?>  
<?import javafx.scene.layout.AnchorPane?>  
  
<AnchorPane   
 maxHeight="-Infinity"   
 maxWidth="-Infinity"   
 minHeight="-Infinity"   
 minWidth="-Infinity"   
 prefHeight="400.0"   
 prefWidth="600.0"   
 xmlns="http://javafx.com/javafx/15.0.1"   
 xmlns:fx="http://javafx.com/fxml/1" >  
 <children>  
 <ComboBox fx:id="comboBox" layoutX="5.0" layoutY="5.0" prefHeight="27.0" prefWidth="100.0" promptText="Select">  
 <items>  
 <FXCollections fx:factory="observableArrayList">  
 </FXCollections>  
 </items>  
 </ComboBox>  
  
 <Button layoutX="120.0" layoutY="5.0" mnemonicParsing="false" onAction="#count" text="Count" />  
 <Button layoutX="185.0" layoutY="5.0" mnemonicParsing="false" onAction="#cancel" text="Cancel" />  
  
 </children>  
</AnchorPane>

CountController.java:

package ui;  
  
import app.\*;  
  
import java.util.\*;  
  
import javafx.fxml.FXML;  
import javafx.scene.control.ComboBox;  
  
public class CountController {  
 private Count mInst;  
 private boolean type;  
  
 @FXML  
 private ComboBox<String> comboBox;  
  
 public CountController(Count mInst, boolean type) {  
 this.mInst = mInst;  
 this.type = type;  
 }  
  
 public void setDataToCombo() {  
 mInst.getMainController();  
 List<IShape> list = MainController.*list*;  
 for (int i = 0; i < list.size(); i++) {  
 comboBox.getItems().add(list.get(i).toString());  
 }  
  
 }  
  
 @FXML  
 private void count() {  
 mInst.getMainController();  
 List<IShape> list = MainController.*list*;  
 for (int i = 0; i < list.size(); i++) {  
 if (list.get(i).toString().equals((String) comboBox.getValue())) {  
 mInst.onDestroy();  
 mInst.getMainController().countByPos(i, type);  
 return;  
 }  
 }  
 }  
  
 @FXML  
 private void cancel() {  
 mInst.onDestroy();  
 }  
}

CrossChecking.java:

package ui;  
  
import java.io.FileInputStream;  
import java.io.IOException;  
  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Scene;  
import javafx.scene.layout.Pane;  
import javafx.stage.Stage;  
  
public class CrossChecking {  
  
 public CrossChecking() { }  
 private Stage stage;  
 private Pane root;  
 private MainController mController;  
 private String fxmlDocPath = "./src/main/java/ui/CrossChecking.fxml";  
  
 public Stage getStage() { return stage; }  
 public Pane getPane() { return root; }  
 public MainController getMainController() { return mController; }  
  
 public void start(MainController mController) throws IOException {  
 this.mController = mController;  
 stage = new Stage();  
 FXMLLoader loader = new FXMLLoader();  
 FileInputStream fxmlStream = new FileInputStream(fxmlDocPath);  
 CrossCheckingController controller = new CrossCheckingController(this);  
 loader.setController(controller);  
 root = (Pane) loader.load(fxmlStream);  
  
 Scene scene = new Scene(root);  
 stage.setScene(scene);  
 stage.setTitle("Check if figures cross");  
 stage.show();  
 controller.setDataToCombo();  
 }  
  
 public void onDestroy() {  
 stage.close();  
 }  
}

CrossChecking.fxml:

<?xml version="1.0" encoding="UTF-8"?>  
  
<?import javafx.collections.\*?>   
<?import javafx.scene.canvas.Canvas?>  
<?import javafx.scene.control.\*?>  
<?import javafx.scene.layout.\*?>  
<?import javafx.scene.image.ImageView?>  
<?import javafx.scene.layout.AnchorPane?>  
  
<AnchorPane   
 maxHeight="-Infinity"   
 maxWidth="-Infinity"   
 minHeight="-Infinity"   
 minWidth="-Infinity"   
 prefHeight="400.0"   
 prefWidth="600.0"   
 xmlns="http://javafx.com/javafx/15.0.1"   
 xmlns:fx="http://javafx.com/fxml/1" >  
 <children>  
 <ComboBox fx:id="comboBox" layoutX="5.0" layoutY="5.0" prefHeight="27.0" prefWidth="100.0" promptText="Select">  
 <items>  
 <FXCollections fx:factory="observableArrayList">  
 </FXCollections>  
 </items>  
 </ComboBox>  
  
 <Button layoutX="120.0" layoutY="5.0" mnemonicParsing="false" onAction="#check" text="Compute" />  
 <Button layoutX="205.0" layoutY="5.0" mnemonicParsing="false" onAction="#cancel" text="Cancel" />  
  
 <ComboBox fx:id="firstFigure" layoutX="5.0" layoutY="50.0" prefHeight="27.0" prefWidth="100.0" promptText="Select">  
 <items>  
 <FXCollections fx:factory="observableArrayList">  
 </FXCollections>  
 </items>  
 </ComboBox>  
  
 <ComboBox fx:id="secondFigure" layoutX="120.0" layoutY="50.0" prefHeight="27.0" prefWidth="100.0" promptText="Select">  
 <items>  
 <FXCollections fx:factory="observableArrayList">  
 </FXCollections>  
 </items>  
 </ComboBox>  
  
 </children>  
</AnchorPane>

CrossCheckingController.java:

package ui;  
  
import app.\*;  
  
import java.util.\*;  
  
import javafx.fxml.FXML;  
import javafx.scene.control.ComboBox;  
  
public class CrossCheckingController {  
 private CrossChecking mInst;  
  
 @FXML  
 private ComboBox<String> comboBox;  
  
 @FXML  
 private ComboBox<String> firstFigure;  
  
 @FXML  
 private ComboBox<String> secondFigure;  
  
 public CrossCheckingController(CrossChecking inst) { mInst = inst; }  
  
 public void setDataToCombo() {  
 comboBox.getItems().clear();  
 comboBox.getItems().addAll(  
 "Circle",  
 "Segment",  
 "TGon",  
 "NGon",  
 "QGon",  
 "Trapeze",  
 "Rectangle",  
 "Polyline"  
 );  
  
 comboBox.setOnAction((event) -> {  
 CrossCheckingController.this.addAttrs((String) comboBox.getValue());  
 });  
 }  
  
 @FXML  
 private void check() {  
 String firstOne = (String) firstFigure.getValue();  
 String secondOne = (String) secondFigure.getValue();  
  
 if (firstOne == null || secondOne == null  
 || firstOne == "" || secondOne == "") {  
 mInst.getMainController().showAlert();  
 return;  
 }  
  
 mInst.getMainController();  
 List<IShape> curList = MainController.*list*;  
 IShape fInstance = null;  
 IShape sInstance = null;  
 for (int i = 0; i < curList.size(); i++) {  
 if (fInstance == null && curList.get(i).toString().equals(firstOne)) {  
 fInstance = curList.get(i);  
 }  
  
 if (sInstance == null && curList.get(i).toString().equals(secondOne)) {  
 sInstance = curList.get(i);  
 }  
 }  
  
 mInst.getMainController().setIfCross(fInstance, sInstance);  
 mInst.onDestroy();  
 }  
  
 @FXML  
 private void cancel() {  
 mInst.onDestroy();  
 }  
  
 private void addAttrs(String type) {  
 firstFigure.getItems().clear();  
 secondFigure.getItems().clear();  
 mInst.getMainController();  
 List<IShape> curList = MainController.*list*;  
 for (int i = 0; i < curList.size(); i++) {  
 if (curList.get(i).getType().equals(type)) {  
 firstFigure.getItems().add(curList.get(i).toString());  
 secondFigure.getItems().add(curList.get(i).toString());  
 }  
 }  
 }  
}

DragFigure.java:

package ui;  
  
import java.io.FileInputStream;  
import java.io.IOException;  
   
import javafx.fxml.FXMLLoader;  
import javafx.scene.Scene;  
import javafx.scene.layout.Pane;  
import javafx.stage.Stage;  
   
public class DragFigure {  
  
 public DragFigure() { }  
 private Stage stage;  
 private Pane root;  
 private MainController mController;  
 private String fxmlDocPath = "./src/main/java/ui/DragFigure.fxml";  
  
 public Stage getStage() { return stage; }  
 public Pane getPane() { return root; }  
 public MainController getMainController() { return mController; }  
  
 public void start(MainController mController) throws IOException {  
 this.mController = mController;  
 stage = new Stage();  
 FXMLLoader loader = new FXMLLoader();  
 FileInputStream fxmlStream = new FileInputStream(fxmlDocPath);  
 DragFigureController controller = new DragFigureController(this);  
 loader.setController(controller);  
 root = (Pane) loader.load(fxmlStream);  
  
 Scene scene = new Scene(root);  
 stage.setScene(scene);  
 stage.setTitle("Drag Figure");  
 stage.show();  
 controller.setDataToCombo();  
 }  
  
 public void onDestroy() {  
 stage.close();  
 }  
}

DragFigure.fxml:

<?xml version="1.0" encoding="UTF-8"?>  
  
<?import javafx.collections.\*?>   
<?import javafx.scene.canvas.Canvas?>  
<?import javafx.scene.control.\*?>  
<?import javafx.scene.layout.\*?>  
<?import javafx.scene.image.ImageView?>  
<?import javafx.scene.layout.AnchorPane?>  
  
<AnchorPane   
 maxHeight="-Infinity"   
 maxWidth="-Infinity"   
 minHeight="-Infinity"   
 minWidth="-Infinity"   
 prefHeight="400.0"   
 prefWidth="600.0"   
 xmlns="http://javafx.com/javafx/15.0.1"   
 xmlns:fx="http://javafx.com/fxml/1" >  
 <children>  
 <ComboBox fx:id="comboBox" layoutX="5.0" layoutY="5.0" prefHeight="27.0" prefWidth="100.0" promptText="Select">  
 <items>  
 <FXCollections fx:factory="observableArrayList">  
 </FXCollections>  
 </items>  
 </ComboBox>  
   
 <ComboBox fx:id="drags" layoutX="5.0" layoutY="50.0" prefHeight="27.0" prefWidth="100.0" promptText="Drag">  
 <items>  
 <FXCollections fx:factory="observableArrayList">  
 </FXCollections>  
 </items>  
 </ComboBox>  
  
 <Button layoutX="120.0" layoutY="5.0" mnemonicParsing="false" onAction="#dragCurrent" text="Drag" />  
 <Button layoutX="180.0" layoutY="5.0" mnemonicParsing="false" onAction="#cancel" text="Cancel" />  
  
 </children>  
</AnchorPane>

DragFigureController.java:

package ui;  
  
import app.\*;  
  
import java.util.\*;  
  
import javafx.fxml.FXML;  
import javafx.scene.control.\*;  
import javafx.scene.control.ComboBox;  
  
public class DragFigureController {  
 private DragFigure mInst;  
 private List<TextField> list = new ArrayList<>();  
 private Spinner<Integer> spinner;  
 private double initX = 5.0;  
 private double initY = 100.0;  
 private double currentX = initX;  
 private double currentY = initY;  
 private final double deltaX = 160.0;  
 private final double deltaY = 30.0;  
 private boolean isLeft = true;  
  
 @FXML  
 private ComboBox<String> comboBox;  
  
 @FXML  
 private ComboBox<String> drags;  
  
 public DragFigureController(DragFigure controller) { mInst = controller; }  
   
 public void setDataToCombo() {  
 comboBox.getItems().clear();  
 mInst.getMainController();  
 List<IShape> curList = MainController.*list*;  
 for (int i = 0; i < curList.size(); i++)  
 comboBox.getItems().add(curList.get(i).toString());  
  
 drags.getItems().clear();  
 currentX = initX;  
 currentY = initY;  
  
 drags.getItems().addAll(  
 "Rot",  
 "SymAxis",  
 "Shift"  
 );  
  
 drags.setOnAction((event) -> {  
 DragFigureController.this.addAttrs((String) drags.getValue());  
 });  
 }  
  
 @FXML  
 private void dragCurrent() {  
 String figure = (String) comboBox.getValue();  
 String type = (String) drags.getValue();  
  
 if (figure == "" || type == ""  
 || figure == null || type == null) {  
 mInst.getMainController().showAlert();  
 return;  
 }  
  
 mInst.getMainController();  
 List<IShape> curList = MainController.*list*;  
 for (int i = 0; i < curList.size(); i++) {  
 if (curList.get(i).toString().equals(figure)) {  
 double[] args;  
 if (type == "SymAxis") {  
 args = new double[1];  
 args[0] = (double) ((Integer) spinner.getValue()).intValue();  
 } else  
 args = new double[list.size()];  
 for (int j = 0; j < list.size(); j++) {  
 args[j] = Double.*parseDouble*(list.get(j).getText());  
 }  
  
 mInst.getMainController().moveFigure(i, type, args);  
 mInst.onDestroy();  
 return;  
 }  
 }  
  
 }  
  
 @FXML  
 private void cancel() {  
 mInst.onDestroy();  
 }  
  
 private void removeSpinner() {  
 if (spinner != null)  
 mInst.getPane().getChildren().remove(spinner);  
 }  
  
 private void addSpinner() {  
 spinner = new Spinner<Integer>();  
 SpinnerValueFactory<Integer> valueFactory =  
 new SpinnerValueFactory.IntegerSpinnerValueFactory(0, 1, 0);  
  
 spinner.setValueFactory(valueFactory);  
  
 spinner.setLayoutX(initX);  
 spinner.setLayoutY(initY);  
  
 mInst.getPane().getChildren().addAll(spinner);  
 }  
  
 private void addAttrs(String value) {  
 for (int i = 0; i < list.size(); i++) {  
 mInst.getPane().getChildren().remove(list.get(i));  
 }  
  
 list = new ArrayList<>();  
 removeSpinner();  
 isLeft = true;  
   
 currentX = initX;  
 currentY = initY;  
  
 if (value == "Rot") {  
 addNFields(1);  
 } else if (value == "SymAxis") {  
 addSpinner();  
 } else {  
 addNFields(2);  
 }  
 }  
  
 private void setCoords(TextField field) {  
 field.setLayoutX(currentX);  
 field.setLayoutY(currentY);  
 if (isLeft) {  
 currentX += deltaX;  
 isLeft = false;  
 } else {  
 currentX -= deltaX;  
 currentY += deltaY;  
 isLeft = true;  
 }  
 }  
  
 private void addField(TextField field) {  
 list.add(field);  
 mInst.getPane().getChildren().addAll(field);  
 }  
  
 private void addNFields(int n) {  
 for (int i = 0; i < n; i++) {  
 TextField field = new TextField();  
 field.setText("0");  
 setCoords(field);  
 addField(field);  
 }  
 }  
}

FigureAddition.java:

package ui;  
  
import java.io.FileInputStream;  
import java.io.IOException;  
   
import javafx.fxml.FXMLLoader;  
import javafx.scene.Scene;  
import javafx.scene.layout.Pane;  
import javafx.stage.Stage;  
   
public class FigureAddition {  
  
 public FigureAddition() { }  
 private Stage stage;  
 private Pane root;  
 private MainController mController;  
 private String fxmlDocPath = "./src/main/java/ui/FigureAddition.fxml";  
  
 public Stage getStage() { return stage; }  
 public Pane getPane() { return root; }  
 public MainController getMainController() { return mController; }  
  
 public void start(MainController mController) throws IOException {  
 this.mController = mController;  
 stage = new Stage();  
 FXMLLoader loader = new FXMLLoader();  
 FileInputStream fxmlStream = new FileInputStream(fxmlDocPath);  
 FigureAdditionController controller = new FigureAdditionController(this);  
 loader.setController(controller);  
 root = (Pane) loader.load(fxmlStream);  
  
 Scene scene = new Scene(root);  
 stage.setScene(scene);  
 stage.setTitle("Add Figures");  
 stage.show();  
 controller.setDataToCombo();  
 }  
  
 public void onDestroy() {  
 stage.close();  
 }  
}

FigureAddition.fxml:

<?xml version="1.0" encoding="UTF-8"?>  
  
<?import javafx.collections.\*?>   
<?import javafx.scene.canvas.Canvas?>  
<?import javafx.scene.control.\*?>  
<?import javafx.scene.layout.\*?>  
<?import javafx.scene.image.ImageView?>  
<?import javafx.scene.layout.AnchorPane?>  
  
<AnchorPane   
 maxHeight="-Infinity"   
 maxWidth="-Infinity"   
 minHeight="-Infinity"   
 minWidth="-Infinity"   
 prefHeight="400.0"   
 prefWidth="600.0"   
 xmlns="http://javafx.com/javafx/15.0.1"   
 xmlns:fx="http://javafx.com/fxml/1" >  
 <children>  
 <ComboBox fx:id="comboBox" layoutX="5.0" layoutY="5.0" prefHeight="27.0" prefWidth="100.0" promptText="Select">  
 <items>  
 <FXCollections fx:factory="observableArrayList">  
 </FXCollections>  
 </items>  
 </ComboBox>  
  
 <Button layoutX="120.0" layoutY="5.0" mnemonicParsing="false" onAction="#addCurrent" text="Add" />  
 <Button layoutX="175.0" layoutY="5.0" mnemonicParsing="false" onAction="#cancel" text="Cancel" />  
  
 </children>  
</AnchorPane>

FigureAdditionController.java:

package ui;  
  
import java.util.\*;  
import java.lang.Exception;  
  
import javafx.fxml.FXML;  
import javafx.scene.control.\*;  
import javafx.scene.control.ComboBox;  
  
public class FigureAdditionController {  
 private FigureAddition mInst;  
 private Spinner<Integer> spinner;  
 private ArrayList<TextField> coords = new ArrayList<TextField>();  
 private double currentX = 5.0;  
 private double currentY = 35.0;  
 private final double deltaX = 160.0;  
 private final double deltaY = 30.0;  
 private boolean isLeft = true;  
  
 @FXML  
 private ComboBox<String> comboBox;  
  
 public FigureAdditionController(FigureAddition inst) { mInst = inst; }  
  
 public void addAttrs(String type) {  
 if (type.equals("Circle")) {  
 addCircle();  
 } else if (type.equals("Segment")) {  
 addSegment();  
 } else if (type.equals("TGon")) {  
 addTGon();  
 } else if (type.equals("NGon")) {  
 addNGon();  
 } else if (type.equals("QGon")) {  
 addQGon();  
 } else if (type.equals("Trapeze")) {  
 addTrapeze();  
 } else if (type.equals("Rectangle")) {  
 addRectangle();  
 } else if (type.equals("Polyline")) {  
 addPolyline();  
 }  
 }  
  
 public void setDataToCombo() {  
 comboBox.getItems().clear();  
 comboBox.getItems().addAll(  
 "Circle",  
 "Segment",  
 "TGon",  
 "NGon",  
 "QGon",  
 "Trapeze",  
 "Rectangle",  
 "Polyline"  
 );  
  
 comboBox.setOnAction((event) -> {  
 if (comboBox.getValue() instanceof String)  
 FigureAdditionController.this.addAttrs((String) comboBox.getValue());  
 });  
 }  
  
 @FXML  
 private void addCurrent() {  
 String type = (String) comboBox.getValue();  
   
 double[] c = new double[coords.size()];  
 for (int i = 0; i < coords.size(); i++) {  
 try {  
 c[i] = Double.*parseDouble*(coords.get(i).getText());  
 } catch (Exception e) {  
 mInst.getMainController().showAlert();  
 e.printStackTrace();  
 return;  
 }  
 }  
 try {  
 mInst.getMainController().addShape(type, c);  
 } catch (Exception e) {  
 mInst.getMainController().showAlert();  
 e.printStackTrace();  
 return;  
 }  
 cancel();  
 }  
  
 @FXML  
 private void cancel() {  
 removeSpinner();  
 clearCoords();  
 mInst.onDestroy();  
 }  
  
 private void removeSpinner() {  
 if (spinner != null)  
 mInst.getPane().getChildren().remove(spinner);  
 }  
  
 private void addSpinner() {  
 spinner = new Spinner<Integer>();  
 SpinnerValueFactory<Integer> valueFactory =  
 new SpinnerValueFactory.IntegerSpinnerValueFactory(1, 5, 1);  
  
 spinner.setValueFactory(valueFactory);  
 spinner.valueProperty().addListener((obs, oldValue, newValue) -> {  
 clearCoords();  
 addNFields(newValue);  
 });  
  
 spinner.setLayoutX(450.0);  
 spinner.setLayoutY(5.0);  
  
 mInst.getPane().getChildren().addAll(spinner);  
 }  
  
 private void clearCoords() {  
 for (int i = 0; i < coords.size(); i++) {  
 if (coords.get(i) != null)  
 mInst.getPane().getChildren().remove(coords.get(i));  
 }  
  
 coords = new ArrayList<TextField>();  
 currentX = 5.0;  
 currentY = 35.0;  
 isLeft = true;  
 }  
  
 private void setCoords(TextField field) {  
 field.setLayoutX(currentX);  
 field.setLayoutY(currentY);  
 if (isLeft) {  
 currentX += deltaX;  
 isLeft = false;  
 } else {  
 currentX -= deltaX;  
 currentY += deltaY;  
 isLeft = true;  
 }  
 }  
  
 private void addField(TextField field) {  
 coords.add(field);  
 mInst.getPane().getChildren().addAll(field);  
 }  
  
 private void addNFields(int n) {  
 for (int i = 0; i < 2 \* n; i++) {  
 TextField field = new TextField();  
 field.setText("0");  
 setCoords(field);  
 addField(field);  
 }  
 }  
  
 private void addCircle() {  
 removeSpinner();  
 clearCoords();  
 addNFields(1);  
 TextField field = new TextField();  
 field.setText("0");  
 setCoords(field);  
 addField(field);  
 }  
  
 private void addSegment() {  
 removeSpinner();  
 clearCoords();  
 addNFields(2);  
 }  
  
 private void addTGon() {  
 removeSpinner();  
 clearCoords();  
 addNFields(3);  
 }  
  
 private void addNGon() {  
 removeSpinner();  
 clearCoords();  
 addSpinner();  
 addNFields(spinner.getValue());  
 }  
  
 private void addQGon() {  
 removeSpinner();  
 clearCoords();  
 addNFields(4);  
 }  
  
 private void addTrapeze() {  
 removeSpinner();  
 addQGon();  
 }  
  
 private void addRectangle() {  
 removeSpinner();  
 addQGon();  
 }  
  
 private void addPolyline() {  
 removeSpinner();  
 clearCoords();  
 addSpinner();  
 addNFields(spinner.getValue());  
 }  
}

FigureRemoval.java:

package ui;  
  
import java.io.FileInputStream;  
import java.io.IOException;  
  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Scene;  
import javafx.scene.layout.Pane;  
import javafx.stage.Stage;  
   
public class FigureRemoval {  
  
 public FigureRemoval() { }  
 private Stage stage;  
 private Pane root;  
 private MainController mController;  
 private String fxmlDocPath = "./src/main/java/ui/FigureRemoval.fxml";  
  
 public Stage getStage() { return stage; }  
 public Pane getPane() { return root; }  
 public MainController getMainController() { return mController; }  
  
 public void start(MainController mController) throws IOException {  
 this.mController = mController;  
 stage = new Stage();  
 FXMLLoader loader = new FXMLLoader();  
 FileInputStream fxmlStream = new FileInputStream(fxmlDocPath);  
 FigureRemovalController controller = new FigureRemovalController(this);  
 loader.setController(controller);  
 root = (Pane) loader.load(fxmlStream);  
  
 Scene scene = new Scene(root);  
 stage.setScene(scene);  
 stage.setTitle("Remove Figure");  
 stage.show();  
 controller.setDataToCombo();  
 }  
  
 public void onDestroy() {  
 stage.close();  
 }  
}

FigureAddition.fxml:

<?xml version="1.0" encoding="UTF-8"?>  
  
<?import javafx.collections.\*?>   
<?import javafx.scene.canvas.Canvas?>  
<?import javafx.scene.control.\*?>  
<?import javafx.scene.layout.\*?>  
<?import javafx.scene.image.ImageView?>  
<?import javafx.scene.layout.AnchorPane?>  
  
<AnchorPane   
 maxHeight="-Infinity"   
 maxWidth="-Infinity"   
 minHeight="-Infinity"   
 minWidth="-Infinity"   
 prefHeight="400.0"   
 prefWidth="600.0"   
 xmlns="http://javafx.com/javafx/15.0.1"   
 xmlns:fx="http://javafx.com/fxml/1" >  
 <children>  
 <ComboBox fx:id="comboBox" layoutX="5.0" layoutY="5.0" prefHeight="27.0" prefWidth="100.0" promptText="Select">  
 <items>  
 <FXCollections fx:factory="observableArrayList">  
 </FXCollections>  
 </items>  
 </ComboBox>  
  
 <Button layoutX="120.0" layoutY="5.0" mnemonicParsing="false" onAction="#addCurrent" text="Add" />  
 <Button layoutX="175.0" layoutY="5.0" mnemonicParsing="false" onAction="#cancel" text="Cancel" />  
  
 </children>  
</AnchorPane>

FigureAdditionController.java:

package ui;  
  
import java.util.\*;  
import java.lang.Exception;  
  
import javafx.fxml.FXML;  
import javafx.scene.control.\*;  
import javafx.scene.control.ComboBox;  
  
public class FigureAdditionController {  
 private FigureAddition mInst;  
 private Spinner<Integer> spinner;  
 private ArrayList<TextField> coords = new ArrayList<TextField>();  
 private double currentX = 5.0;  
 private double currentY = 35.0;  
 private final double deltaX = 160.0;  
 private final double deltaY = 30.0;  
 private boolean isLeft = true;  
  
 @FXML  
 private ComboBox<String> comboBox;  
  
 public FigureAdditionController(FigureAddition inst) { mInst = inst; }  
  
 public void addAttrs(String type) {  
 if (type.equals("Circle")) {  
 addCircle();  
 } else if (type.equals("Segment")) {  
 addSegment();  
 } else if (type.equals("TGon")) {  
 addTGon();  
 } else if (type.equals("NGon")) {  
 addNGon();  
 } else if (type.equals("QGon")) {  
 addQGon();  
 } else if (type.equals("Trapeze")) {  
 addTrapeze();  
 } else if (type.equals("Rectangle")) {  
 addRectangle();  
 } else if (type.equals("Polyline")) {  
 addPolyline();  
 }  
 }  
  
 public void setDataToCombo() {  
 comboBox.getItems().clear();  
 comboBox.getItems().addAll(  
 "Circle",  
 "Segment",  
 "TGon",  
 "NGon",  
 "QGon",  
 "Trapeze",  
 "Rectangle",  
 "Polyline"  
 );  
  
 comboBox.setOnAction((event) -> {  
 if (comboBox.getValue() instanceof String)  
 FigureAdditionController.this.addAttrs((String) comboBox.getValue());  
 });  
 }  
  
 @FXML  
 private void addCurrent() {  
 String type = (String) comboBox.getValue();  
   
 double[] c = new double[coords.size()];  
 for (int i = 0; i < coords.size(); i++) {  
 try {  
 c[i] = Double.*parseDouble*(coords.get(i).getText());  
 } catch (Exception e) {  
 mInst.getMainController().showAlert();  
 e.printStackTrace();  
 return;  
 }  
 }  
 try {  
 mInst.getMainController().addShape(type, c);  
 } catch (Exception e) {  
 mInst.getMainController().showAlert();  
 e.printStackTrace();  
 return;  
 }  
 cancel();  
 }  
  
 @FXML  
 private void cancel() {  
 removeSpinner();  
 clearCoords();  
 mInst.onDestroy();  
 }  
  
 private void removeSpinner() {  
 if (spinner != null)  
 mInst.getPane().getChildren().remove(spinner);  
 }  
  
 private void addSpinner() {  
 spinner = new Spinner<Integer>();  
 SpinnerValueFactory<Integer> valueFactory =  
 new SpinnerValueFactory.IntegerSpinnerValueFactory(1, 5, 1);  
  
 spinner.setValueFactory(valueFactory);  
 spinner.valueProperty().addListener((obs, oldValue, newValue) -> {  
 clearCoords();  
 addNFields(newValue);  
 });  
  
 spinner.setLayoutX(450.0);  
 spinner.setLayoutY(5.0);  
  
 mInst.getPane().getChildren().addAll(spinner);  
 }  
  
 private void clearCoords() {  
 for (int i = 0; i < coords.size(); i++) {  
 if (coords.get(i) != null)  
 mInst.getPane().getChildren().remove(coords.get(i));  
 }  
  
 coords = new ArrayList<TextField>();  
 currentX = 5.0;  
 currentY = 35.0;  
 isLeft = true;  
 }  
  
 private void setCoords(TextField field) {  
 field.setLayoutX(currentX);  
 field.setLayoutY(currentY);  
 if (isLeft) {  
 currentX += deltaX;  
 isLeft = false;  
 } else {  
 currentX -= deltaX;  
 currentY += deltaY;  
 isLeft = true;  
 }  
 }  
  
 private void addField(TextField field) {  
 coords.add(field);  
 mInst.getPane().getChildren().addAll(field);  
 }  
  
 private void addNFields(int n) {  
 for (int i = 0; i < 2 \* n; i++) {  
 TextField field = new TextField();  
 field.setText("0");  
 setCoords(field);  
 addField(field);  
 }  
 }  
  
 private void addCircle() {  
 removeSpinner();  
 clearCoords();  
 addNFields(1);  
 TextField field = new TextField();  
 field.setText("0");  
 setCoords(field);  
 addField(field);  
 }  
  
 private void addSegment() {  
 removeSpinner();  
 clearCoords();  
 addNFields(2);  
 }  
  
 private void addTGon() {  
 removeSpinner();  
 clearCoords();  
 addNFields(3);  
 }  
  
 private void addNGon() {  
 removeSpinner();  
 clearCoords();  
 addSpinner();  
 addNFields(spinner.getValue());  
 }  
  
 private void addQGon() {  
 removeSpinner();  
 clearCoords();  
 addNFields(4);  
 }  
  
 private void addTrapeze() {  
 removeSpinner();  
 addQGon();  
 }  
  
 private void addRectangle() {  
 removeSpinner();  
 addQGon();  
 }  
  
 private void addPolyline() {  
 removeSpinner();  
 clearCoords();  
 addSpinner();  
 addNFields(spinner.getValue());  
 }  
}

FigureRemoval.java:

package ui;  
  
import java.io.FileInputStream;  
import java.io.IOException;  
  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Scene;  
import javafx.scene.layout.Pane;  
import javafx.stage.Stage;  
   
public class FigureRemoval {  
  
 public FigureRemoval() { }  
 private Stage stage;  
 private Pane root;  
 private MainController mController;  
 private String fxmlDocPath = "./src/main/java/ui/FigureRemoval.fxml";  
  
 public Stage getStage() { return stage; }  
 public Pane getPane() { return root; }  
 public MainController getMainController() { return mController; }  
  
 public void start(MainController mController) throws IOException {  
 this.mController = mController;  
 stage = new Stage();  
 FXMLLoader loader = new FXMLLoader();  
 FileInputStream fxmlStream = new FileInputStream(fxmlDocPath);  
 FigureRemovalController controller = new FigureRemovalController(this);  
 loader.setController(controller);  
 root = (Pane) loader.load(fxmlStream);  
  
 Scene scene = new Scene(root);  
 stage.setScene(scene);  
 stage.setTitle("Remove Figure");  
 stage.show();  
 controller.setDataToCombo();  
 }  
  
 public void onDestroy() {  
 stage.close();  
 }  
}

FigureRemoval.fxml:

<?xml version="1.0" encoding="UTF-8"?>  
  
<?import javafx.collections.\*?>   
<?import javafx.scene.canvas.Canvas?>  
<?import javafx.scene.control.\*?>  
<?import javafx.scene.layout.\*?>  
<?import javafx.scene.image.ImageView?>  
<?import javafx.scene.layout.AnchorPane?>  
  
<AnchorPane   
 maxHeight="-Infinity"   
 maxWidth="-Infinity"   
 minHeight="-Infinity"   
 minWidth="-Infinity"   
 prefHeight="400.0"   
 prefWidth="600.0"   
 xmlns="http://javafx.com/javafx/15.0.1"   
 xmlns:fx="http://javafx.com/fxml/1" >  
 <children>  
 <ComboBox fx:id="comboBox" layoutX="5.0" layoutY="5.0" prefHeight="27.0" prefWidth="100.0" promptText="Select">  
 <items>  
 <FXCollections fx:factory="observableArrayList">  
 </FXCollections>  
 </items>  
 </ComboBox>  
  
 <Button layoutX="120.0" layoutY="5.0" mnemonicParsing="false" onAction="#removeCurrent" text="Remove" />  
 <Button layoutX="200.0" layoutY="5.0" mnemonicParsing="false" onAction="#cancel" text="Cancel" />  
  
 </children>  
</AnchorPane>

FigureRemovalController.java:

package ui;  
  
import app.\*;  
  
import java.util.\*;  
  
import javafx.fxml.FXML;  
import javafx.scene.control.ComboBox;  
public class FigureRemovalController {  
 private FigureRemoval mInst;  
  
 @FXML  
 private ComboBox<String> comboBox;  
  
 public FigureRemovalController(FigureRemoval inst) { mInst = inst; }  
  
 public void setDataToCombo() {  
 comboBox.getItems().clear();  
 mInst.getMainController();  
 List<IShape> curList = MainController.*list*;  
 for (int i = 0; i < curList.size(); i++) {  
 comboBox.getItems().add(curList.get(i).toString());  
 }  
 }  
  
 @FXML  
 private void removeCurrent() {  
 mInst.getMainController();  
 List<IShape> curList = MainController.*list*;  
 String obj = (String) comboBox.getValue();  
 for (int i = 0; i < curList.size(); i++) {  
 if (curList.get(i).toString().equals(obj)) {  
 mInst.getMainController().removeFigureByPosition(i);  
 mInst.onDestroy();  
 return;  
 }  
 }  
 }  
  
 @FXML  
 private void cancel() {  
 mInst.onDestroy();  
 }  
}

Main.java:

package ui;  
  
import java.io.FileInputStream;  
import java.io.IOException;  
  
import com.mongodb.ConnectionString;  
import com.mongodb.MongoClientSettings;  
import com.mongodb.client.MongoClient;  
import com.mongodb.client.MongoClients;  
  
import javafx.application.Application;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Scene;  
import javafx.scene.layout.Pane;  
import javafx.stage.Stage;  
import ui.repository.MongoDb;  
   
public class Main extends Application {  
 public static void main(String[] args) {  
 Application.*launch*(args);  
 }  
  
 private Pane root;  
 private MongoDb mongodb = null;  
 private MongoClient client = null;  
 private MongoClientSettings mongoSettings = null;  
 private final ConnectionString mongoClientUri = new ConnectionString("mongodb+srv://maks:geometrynoname@javageometryclustrer.yzcxxso.mongodb.net/?retryWrites=true&w=majority");  
  
 public Pane getPane() { return root; }  
 public MongoDb getDb() { return mongodb; }  
  
 @Override  
 public void start(Stage stage) throws IOException {  
 FXMLLoader loader = new FXMLLoader();  
 String fxmlDocPath = "./src/main/java/ui/Main.fxml";  
 FileInputStream fxmlStream = new FileInputStream(fxmlDocPath);  
 mongoSettings = MongoClientSettings.*builder*()  
 .applyConnectionString(mongoClientUri)  
 .build();  
 client = MongoClients.*create*(mongoSettings);  
 mongodb = MongoDb.*create*(client, null);  
   
 loader.setController(new MainController(this));  
 root = (Pane) loader.load(fxmlStream);  
   
 Scene scene = new Scene(root);  
 stage.setScene(scene);  
 stage.setTitle("Geometry");  
 stage.show();  
 }  
  
 @Override  
 public void stop() {  
 client.close();  
 client = null;  
 mongoSettings = null;  
 MongoDb.*clear*();  
 mongodb = null;  
 }  
}

Main.fxml:

<?xml version="1.0" encoding="UTF-8"?>  
  
<?import javafx.scene.canvas.Canvas?>  
<?import javafx.scene.control.Button?>  
<?import javafx.scene.image.ImageView?>  
<?import javafx.scene.layout.AnchorPane?>  
  
<AnchorPane   
 maxHeight="-Infinity"   
 maxWidth="-Infinity"   
 minHeight="-Infinity"   
 minWidth="-Infinity"   
 prefHeight="400.0"   
 prefWidth="630.0"   
 xmlns="http://javafx.com/javafx/15.0.1"   
 xmlns:fx="http://javafx.com/fxml/1" >  
 <children>  
 <Canvas fx:id="canvas" height="400.0" layoutX="0.0" layoutY="0.0" width="400.0" />  
 </children>  
  
 <children>  
 <Button layoutX="450.0" layoutY="5.0" mnemonicParsing="false" onAction="#addFigure" text="Add Figures" />  
 <!-- <ImageView fx:id="image" fitHeight="150.0" fitWidth="200.0" layoutX="386.0" layoutY="50.0" pickOnBounds="true" preserveRatio="true" /> -->  
 <!-- <Button layoutX="450.0" layoutY="212.0" mnemonicParsing="false" onAction="#saveImage" text="Перенос изображния" /> -->  
 <Button layoutX="450.0" layoutY="35.0" mnemonicParsing="false" onAction="#saveToFile" text="Save To File" />  
 <Button layoutX="450.0" layoutY="65.0" mnemonicParsing="false" onAction="#uploadFromFile" text="Upload From File" />  
 <Button layoutX="450.0" layoutY="95.0" mnemonicParsing="false" onAction="#saveAsImg" text="Save As Image" />  
 <Button layoutX="450.0" layoutY="125.0" mnemonicParsing="false" onAction="#dragFigure" text="Drag Figure" />  
 <Button layoutX="450.0" layoutY="155.0" mnemonicParsing="false" onAction="#removeFigure" text="Remove Figure" />  
 <Button layoutX="450.0" layoutY="185.0" mnemonicParsing="false" onAction="#clear" text="Clear" />  
 <Button layoutX="450.0" layoutY="215.0" mnemonicParsing="false" onAction="#computeS" text="Compute Square" />  
 <Button layoutX="450.0" layoutY="245.0" mnemonicParsing="false" onAction="#computeP" text="Compute Perimeter" />  
 <Button layoutX="450.0" layoutY="275.0" mnemonicParsing="false" onAction="#checkIfCross" text="Check if cross" />  
 <Button layoutX="450.0" layoutY="305.0" mnemonicParsing="false" onAction="#addToDatabase" text="Add figures to db" />  
 <Button layoutX="450.0" layoutY="335.0" mnemonicParsing="false" onAction="#retrieveFromDatabase" text="Get figures from db" />  
 </children>  
</AnchorPane>

MainController.java:

private final String fileName = "./src/main/java/ui/figures.txt";  
private final String pngFilePath = "./src/main/java/ui/snapshot.png";  
private Main mInst;  
  
private final int answerLayoutX = 450;  
private final int answerLayoutY = 305;  
private TextField answer = null;  
  
@FXML  
 private Canvas canvas;  
  
public static List<IShape> *list* = new ArrayList<>();  
public static List<String> *types* = new ArrayList<>();  
  
public void showAlert() {  
 Alert alert = new Alert(AlertType.*INFORMATION*);  
 alert.setTitle("Error alert!");  
  
 // Header Text: null  
 alert.setHeaderText(null);  
 alert.setContentText("Error occured, check stack trace.");  
  
 alert.showAndWait();  
 }  
  
public void moveFigure(int pos, String type, double[] args) {  
 *log*.severe(String.*format*("Move figure: %s", type));  
 try {  
 switch (type) {  
 case "Rot":  
 *list*.get(pos).rot(args[0]);  
 break;  
 case "SymAxis":  
 *list*.get(pos).symAxis((int)args[0]);  
 break;  
 case "Shift":  
 *list*.get(pos).shift(new Point2D(args[0], args[1]));  
 break;  
 }  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
  
 clearCanvasAndDrawAll(canvas.getGraphicsContext2D(), new IShape[] {*list*.get(pos)});  
}  
  
// callback func  
public void countByPos(int pos, boolean type) {  
 *log*.severe("Count perimeter or square by position");  
 clearCanvasAndDrawAll(canvas.getGraphicsContext2D(), new IShape[] {*list*.get(pos)});  
  
 answer = new TextField();  
 answer.setLayoutX(answerLayoutX);  
 answer.setLayoutY(answerLayoutY);  
 double res = 0.0;  
 try {  
 if (!type)  
 res = *list*.get(pos).length();  
 else  
 res = *list*.get(pos).square();  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
  
 answer.setText(String.*valueOf*(res));  
 mInst.getPane().getChildren().addAll(answer);  
}  
  
// callback func  
public void setIfCross(IShape first, IShape second) {  
 *log*.severe("Check if cross");  
 clearCanvasAndDrawAll(canvas.getGraphicsContext2D(), new IShape[] {first, second});  
  
 answer = new TextField();  
 answer.setLayoutX(answerLayoutX);  
 answer.setLayoutY(answerLayoutY);  
 try {  
 if (first.cross(second))  
 answer.setText("Cross");  
 else  
 answer.setText("Not Cross");  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
  
 mInst.getPane().getChildren().addAll(answer);  
}  
  
// callback func  
public void removeFigureByPosition(int pos) {  
 *log*.severe("Remove figure");  
 *list*.remove(pos);  
 clearCanvasAndDrawAll(canvas.getGraphicsContext2D(), null);  
}  
  
public void addCircle(double[] coords) throws Exception {  
 *log*.severe("Add Circle");  
 double r = coords[2];  
 Point2D point = new Point2D(coords[0], coords[1]);  
 addToListAndDraw(new Circle(point, r));  
}  
  
public void addNGon(double[] coords) throws Exception {  
 *log*.severe("Add NGon");  
 int n = coords.length / 2;  
 Point2D[] points = new Point2D[n];  
 for (int i = 0; i < coords.length; i += 2) {  
 points[i / 2] = new Point2D(coords[i], coords[i + 1]);  
 }  
  
 addToListAndDraw(new NGon(points));  
}  
  
public void addQGon(double[] coords) throws Exception {  
 *log*.severe("Add QGon");  
 Point2D[] points = new Point2D[4];  
 for (int i = 0; i < coords.length; i += 2) {  
 points[i / 2] = new Point2D(coords[i], coords[i + 1]);  
 }  
 addToListAndDraw(new QGon(points));   
}  
  
public void addRectangle(double[] coords) throws Exception {  
 *log*.severe("Add rectanle");  
 Point2D[] points = new Point2D[4];  
 for (int i = 0; i < coords.length; i += 2) {  
 points[i / 2] = new Point2D(coords[i], coords[i + 1]);  
 }  
 addToListAndDraw(new Rectangle(points));  
}  
  
public void addTrapeze(double[] coords) throws Exception {  
 *log*.severe("Add trapeze");  
 Point2D[] points = new Point2D[4];  
 for (int i = 0; i < coords.length; i += 2) {  
 points[i / 2] = new Point2D(coords[i], coords[i + 1]);  
 }  
 addToListAndDraw(new Trapeze(points));  
}  
  
public void addPolyline(double[] coords) throws Exception {  
 *log*.severe("Add polyline");  
 int n = coords.length / 2;  
 Point2D[] points = new Point2D[n];  
 for (int i = 0; i < coords.length; i += 2) {  
 points[i / 2] = new Point2D(coords[i], coords[i + 1]);  
 }  
 addToListAndDraw(new Polyline(points));  
}  
  
public void addTGon(double[] coords) throws Exception {  
 *log*.severe("Add TGon");  
 int n = 3;  
 Point2D[] points = new Point2D[n];  
 for (int i = 0; i < coords.length; i += 2) {  
 points[i / 2] = new Point2D(coords[i], coords[i + 1]);  
 }  
  
 addToListAndDraw(new TGon(points));  
}  
  
public void addSegment(double[] coords) throws Exception {  
 *log*.severe("Add segment");  
 addToListAndDraw(new Segment(new Point2D(coords[0], coords[1]),  
 new Point2D(coords[2], coords[3])));  
}  
  
public void addShape(String type, double[] coords) throws Exception {  
 if (type.equals("Circle")) {  
 addCircle(coords);  
 } else if (type.equals("Segment")) {  
 addSegment(coords);  
 } else if (type.equals("TGon")) {  
 addTGon(coords);  
 } else if (type.equals("NGon")) {  
 addNGon(coords);  
 } else if (type.equals("QGon")) {  
 addQGon(coords);  
 } else if (type.equals("Trapeze")) {  
 addTrapeze(coords);  
 } else if (type.equals("Rectangle")) {  
 addRectangle(coords);  
 } else if (type.equals("Polyline")) {  
 addPolyline(coords);  
 }  
}  
   
@FXML  
private void initialize()   
{  
 *log*.severe("Clearing canvas");  
 clearCanvas(canvas.getGraphicsContext2D());  
}  
  
public MainController(Main mInst) { this.mInst = mInst; }  
  
@FXML  
private void addFigure() {  
 FigureAddition addition = new FigureAddition();  
 try {  
 addition.start(this);  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
}  
  
@FXML  
private void saveToFile() {  
 *log*.severe("Save to text file");  
 writeToFile();   
}  
  
@FXML  
private void uploadFromFile() {  
 *log*.severe("Upload from text file");  
 File file = new File("./src/main/java/ui/figures.txt");  
 FileInputStream fstream = null;  
 try {  
 fstream = new FileInputStream(file);  
 } catch (FileNotFoundException e) {  
 e.printStackTrace();  
 return;  
 }  
  
 BufferedReader br = new BufferedReader(new InputStreamReader(fstream));  
 try {  
 String line;  
 int lineInd = 0;  
 while ((line = br.readLine()) != null) {  
 IShape restoredShape = IfigureFactory.*create*(line);  
 if (restoredShape != null) {  
 addToListAndDraw(restoredShape);  
 } else {  
 System.*out*.println(String.*format*("Shape on line %d is invalid, ignoring it", lineInd));  
 }  
 lineInd++;  
 }  
 } catch (Exception e) {  
 showAlert();  
 e.printStackTrace();  
 }  
 try {  
 br.close();  
 } catch (IOException e) {  
 showAlert();  
 e.printStackTrace();  
 }  
}  
  
@FXML  
private void saveAsImg() {  
 *log*.severe("Save as image");  
 SnapshotParameters sp = new SnapshotParameters();  
 sp.setFill(Color.*TRANSPARENT*);  
 WritableImage wi = new WritableImage((int)(canvas.getWidth()), (int)(canvas.getHeight()));  
 Image img = canvas.snapshot(null, wi);  
 File file = new File(pngFilePath);  
 BufferedImage bufImg = SwingFXUtils.*fromFXImage*(img, null);  
 try {  
 ImageIO.*write*(bufImg, "png", file);  
 } catch (IOException e) {  
 e.printStackTrace();  
 showAlert();  
 }  
}  
  
@FXML  
private void dragFigure() {  
 DragFigure dragger = new DragFigure();  
 try {  
 dragger.start(this);  
 } catch (Exception e) {  
 e.printStackTrace();  
 showAlert();  
 }  
}  
  
@FXML  
private void removeFigure() {  
 FigureRemoval removal = new FigureRemoval();  
 try {  
 removal.start(this);  
 } catch (Exception e) {  
 e.printStackTrace();  
 showAlert();  
 }  
}  
  
@FXML  
private void clear() {  
 removeAll();  
}  
  
@FXML  
private void computeS() {  
 Count cnt = new Count();  
 try {  
 cnt.start(this, true);  
 } catch (Exception e) {  
 e.printStackTrace();  
 showAlert();  
 }  
}  
  
@FXML  
private void computeP() {  
 Count cnt = new Count();  
 try {  
 cnt.start(this, false);  
 } catch (Exception e) {  
 e.printStackTrace();  
 showAlert();  
 }  
}  
  
@FXML  
private void checkIfCross() {  
 CrossChecking checking = new CrossChecking();  
 try {  
 checking.start(this);  
 } catch (Exception e) {  
 e.printStackTrace();  
 showAlert();  
 }  
}  
  
@FXML  
private void addToDatabase() {  
 *log*.severe("Add to database");  
 MongoCollection<Document> mongoCollection = mInst.getDb().getCollection(null);  
 mongoCollection.drop();  
  
 *list*.forEach((shape) -> {  
 mongoCollection.insertOne(shape.toBson());  
 });  
}  
  
@FXML  
private void retrieveFromDatabase() {  
 *log*.severe("Retrieve from database");  
 MongoCollection<Document> mongoCollection = mInst.getDb().getCollection(null);  
 mongoCollection.find().forEach((document) -> {  
 Object data = document.get("data");  
 if (data instanceof String) {  
 String buildString = (String)data;  
 IShape restoredShape = IfigureFactory.*create*(buildString);  
 if (restoredShape != null) {  
 addToListAndDraw(restoredShape);  
 } else {  
 System.*out*.println("Data is invalid");  
 }  
 }  
 });  
}  
  
private void clearCanvas(GraphicsContext gc) {  
 gc.clearRect(0, 0, canvas.getWidth(), canvas.getHeight());  
 gc.setStroke(Color.*GREY*);  
 gc.strokeLine(200.0, 0.0, 200.0, 400.0);  
 gc.strokeLine(0.0, 200.0, 400.0, 200.0);  
 gc.setStroke(Color.*BLACK*);  
}  
  
private boolean findIn(IShape shape, IShape[] shapeList) {  
 if (shapeList == null || shape == null)  
 return false;  
  
 for (int i = 0; i < shapeList.length; i++) {  
 if (shape.toString().equals(shapeList[i].toString()))  
 return true;  
 }  
  
 return false;  
}  
  
private void clearCanvasAndDrawAll(GraphicsContext gc, IShape[] redList) {  
 clearCanvas(gc);  
  
 *list*.forEach((shape) -> {  
 try {  
 if (findIn(shape, redList)) {  
 gc.setStroke(Color.*RED*);  
 DrawerFactory.*create*(shape).draw(shape, gc);  
 gc.setStroke(Color.*BLACK*);  
 } else {  
 DrawerFactory.*create*(shape).draw(shape, gc);  
 }  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 });  
}  
  
private void writeToFile() {  
 try {  
 BufferedWriter writer = new BufferedWriter(new FileWriter(fileName));  
 String endOf = "\n";  
 for (int i = 0; i < *list*.size(); i++) {  
 if (i == *list*.size() - 1)  
 endOf = "";  
 writer.write(*list*.get(i).toString() + endOf);  
 }  
 writer.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 showAlert();  
 }  
}  
  
private void addToListAndDraw(IShape shape) {  
 *list*.add(shape);  
 try {  
 DrawerFactory.*create*(shape).draw(shape, canvas.getGraphicsContext2D());  
 } catch (Exception e) {  
 e.printStackTrace();  
 showAlert();  
 }  
}  
  
private void removeAll() {  
 *list* = new ArrayList<IShape>();  
 clearCanvas(canvas.getGraphicsContext2D());  
}

ConstructorHolder.java:

package ui.deserializers;  
  
import java.util.ArrayList;  
import java.util.HashMap;  
  
public final class ConstructorHolder {  
 private String name = null;  
 private HashMap<String, String[]> args = new HashMap<>();  
  
 private ConstructorHolder(String name, HashMap<String, String[]> args) {  
 this.name = name;  
 this.args = args;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public HashMap<String, String[]> getArgs() {  
 return args;  
 }  
   
 public static ConstructorHolder parseFromString(String buildString) throws Exception {  
 String[] buildRes = new String[2];  
  
 parsing:  
 for (int i = 0; i < buildString.length(); i++) {  
 char c = buildString.charAt(i);  
 if (c == '(') {  
 for (int j = buildString.length() - 1; j >= i; j--) {  
 char t = buildString.charAt(j);  
 if (t == ')') {  
 buildRes[0] = buildString.substring(0, i);  
 buildRes[1] = buildString.substring(i + 1, j);  
 break parsing;  
 }  
 }  
 }  
 }  
 String name = buildRes[0];  
 String args = buildRes[1];  
 return new ConstructorHolder(name, *parseArgs*(args));  
 }  
  
 private static HashMap<String, String[]> parseArgs(String strArgs) {  
 HashMap<String, String[]> args = new HashMap<>();  
  
 int lastInd = 0;  
 int bracketsBalance = 0;  
 int state = 0; // 0 - бежим по имени аргумента, 1 - бежим по значению аргумента, 2 - бежим по внутренней кухне аргумента,  
 String curArgName = null;  
 for (int i = 0; i <= strArgs.length(); i++) {  
 if (i >= strArgs.length()) {  
 args.put(curArgName, *parseArrayArgs*(strArgs.substring(lastInd, i)));  
 break;  
 }  
 char curChar = strArgs.charAt(i);  
 if (state == 0) {  
 if (curChar != '=') {  
 continue;  
 }  
 curArgName = strArgs.substring(lastInd, i);  
 state = 1;  
 lastInd = i + 1;  
 } else if (state == 1) {  
 if (curChar == '(') {  
 bracketsBalance++;  
 state = 2;  
 continue;  
 }  
  
 if (curChar == ',') {  
 args.put(curArgName, *parseArrayArgs*(strArgs.substring(lastInd, i)));  
 curArgName = null;  
 state = 0;  
 lastInd = i + 2;  
 i++;  
 continue;  
 }  
 } else if (state == 2) {  
 if (curChar == '(')  
 bracketsBalance++;  
  
 if (curChar == ')' && --bracketsBalance == 0) {  
 state = 1;  
 continue;  
 }  
 }  
 }  
  
 return args;  
 }  
  
 private static String[] parseArrayArgs(String arrString) {  
 if (arrString == null || arrString.length() == 0 || arrString.charAt(0) != '(') {  
 return new String[] { arrString };  
 }  
  
 ArrayList<String> list = new ArrayList<>();  
  
 int lastInd = 1;  
 int bracketBalance = 0;  
 for (int i = 1; i < arrString.length(); i++) {  
 if (i == arrString.length() - 1) {  
 list.add(arrString.substring(lastInd, i));  
 break;  
 }  
 char cur = arrString.charAt(i);  
 switch (cur) {  
 case '(':  
 bracketBalance++;  
 break;  
 case ')':  
 bracketBalance--;  
 break;  
 case ',':  
 if (bracketBalance != 0)  
 continue;  
 list.add(arrString.substring(lastInd, i));  
 lastInd = i + 2;  
 i++;  
 }  
 }  
  
 return list.toArray(new String[0]);  
 }  
}

IfigureFactory.java:

package ui.deserializers;  
  
import java.util.HashMap;  
  
import app.Circle;  
import app.IShape;  
import app.NGon;  
import app.Point2D;  
import app.Polyline;  
import app.QGon;  
import app.Rectangle;  
import app.Segment;  
import app.TGon;  
import app.Trapeze;  
  
public class IfigureFactory {  
   
 public static IShape create(String buildString) {  
 IShape shape = null;  
 try {  
 ConstructorHolder holder = ConstructorHolder.*parseFromString*(buildString);  
 shape = *createByName*(holder.getName(), holder.getArgs());  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
  
 return shape;  
 }  
  
 private static Point2D createPoint(HashMap<String, String[]> args) throws Exception {  
 Point2D point = null;  
   
 if (args.containsKey("dims") && args.containsKey("x")) {  
 Integer dims = Integer.*parseInt*(args.get("dims")[0]);  
 String[] coords = args.get("x");  
  
 if (coords.length != dims || dims != 2) {  
 throw new Exception("Dims not valid");  
 }  
  
 double[] intCoords = new double[dims];  
 for (int i = 0; i < coords.length; i++) {  
 intCoords[i] = (double)Double.*parseDouble*(coords[i]);  
 }  
  
 point = new Point2D(intCoords);  
 }  
  
 return point;  
 }  
  
 private interface Creator<T> {  
 T create(Point2D[] points) throws Exception;  
 }  
  
 private static <T> T createNShape(HashMap<String, String[]> args, Creator<T> creator) throws Exception {  
 T ngon = null;  
  
 if (args.containsKey("n") && args.containsKey("p")) {  
 Integer n = Integer.*parseInt*(args.get("n")[0]);  
 String[] coords = args.get("p");  
  
 if (coords.length != n) {  
 throw new Exception("NGon: n not valid");  
 }  
  
 Point2D[] points = new Point2D[n];  
 for (int i = 0; i < coords.length; i++) {  
 ConstructorHolder holder = ConstructorHolder.*parseFromString*(coords[i]);  
 points[i] = *createPoint*(holder.getArgs());  
 }  
  
 ngon = creator.create(points);  
 }  
  
 return ngon;  
 }  
  
 private static Circle createCircle(HashMap<String, String[]> args) throws Exception {  
 Circle circle = null;  
  
 if (args.containsKey("p") && args.containsKey("r")) {  
 Double r = Double.*parseDouble*(args.get("r")[0]);  
 String center = args.get("p")[0];  
 Point2D point = null;  
  
 ConstructorHolder holder = ConstructorHolder.*parseFromString*(center);  
 point = *createPoint*(holder.getArgs());  
 circle = new Circle(point, r);  
 }  
  
 return circle;  
 }  
  
 private static Segment createSegment(HashMap<String, String[]> args) throws Exception {  
 Segment segment = null;  
  
 if (args.containsKey("start") && args.containsKey("finish")) {  
 String start = args.get("start")[0];  
 String finish = args.get("finish")[0];  
  
 Point2D startP = null;  
 Point2D finishP = null;  
 ConstructorHolder startHolder = ConstructorHolder.*parseFromString*(start);  
 ConstructorHolder finishHolder = ConstructorHolder.*parseFromString*(finish);  
  
 startP = *createPoint*(startHolder.getArgs());  
 finishP = *createPoint*(finishHolder.getArgs());  
  
 segment = new Segment(startP, finishP);  
 }  
  
 return segment;  
 }  
  
 private static IShape createByName(String name, HashMap<String, String[]> args) throws Exception {  
 switch(name){  
 case "NGon":  
 return *createNShape*(args, (p) -> new NGon(p));  
 case "Polyline":  
 return *createNShape*(args, (p) -> new Polyline(p));  
 case "QGon":  
 return *createNShape*(args, (p) -> new QGon(p));  
 case "Rectangle":  
 return *createNShape*(args, (p) -> new Rectangle(p));  
 case "Segment":  
 return *createSegment*(args);  
 case "TGon":  
 return *createNShape*(args, (p) -> new TGon(p));  
 case "Trapeze":  
 return *createNShape*(args, (p) -> new Trapeze(p));  
 case "Circle":  
 return *createCircle*(args);  
 default:  
 return null;  
 }  
 };  
  
}

CircleDrawer.java:

package ui.drawers;  
  
import app.IShape;  
import app.Circle;  
  
import javafx.scene.canvas.GraphicsContext;  
  
public final class CircleDrawer implements Drawer {  
   
 private static CircleDrawer *drawer* = null;  
 private final String className = Circle.class.getName();  
  
 @Override  
 public String getClassName() {  
 return className;  
 }  
  
 @Override  
 public void draw(IShape shape, GraphicsContext gc) throws Exception {  
 Circle circle = (Circle) shape;  
 gc.strokeOval(  
 circle.getP().getX(0) - circle.getR() + *centerX*,  
 circle.getP().getX(1) - circle.getR() + *centerY*,  
 circle.getR() \* 2,  
 circle.getR() \* 2  
 );  
 }  
  
 public static CircleDrawer createInstance() {  
 if (*drawer* == null)  
 *drawer* = new CircleDrawer();  
   
 return *drawer*;  
 }  
}

Drawer.java:

package ui.drawers;  
  
import app.IShape;  
  
import javafx.scene.canvas.GraphicsContext;  
  
public interface Drawer {  
   
 static double *centerX* = 200;  
 static double *centerY* = 200;  
  
 public String getClassName();  
  
 public void draw(IShape shape, GraphicsContext gc) throws Exception;  
}

DrawerFactory.java:

package ui.drawers;  
  
import app.IShape;  
  
import java.util.ArrayList;  
  
public class DrawerFactory {  
 public static Drawer create(IShape shape) throws Exception {  
 for (Drawer drawer: *drawerDelegates*) {  
 if (drawer.getClassName() == shape.getClass().getName()) {  
 return drawer;  
 }  
 }  
  
 throw new Exception("Appropriate drawer is not implemented");  
 }  
  
 private static ArrayList<Drawer> *drawerDelegates* = new ArrayList<>() {{  
 add(CircleDrawer.*createInstance*());  
 add(NGonDrawer.*createInstance*());  
 add(PolylineDrawer.*createInstance*());  
 add(QGonDrawer.*createInstance*());  
 add(RectangleDrawer.*createInstance*());  
 add(SegmentDrawer.*createInstance*());  
 add(TGonDrawer.*createInstance*());  
 add(TrapezeDrawer.*createInstance*());  
 }};  
}

NGonDrawer.java:

package ui.drawers;  
  
import app.IShape;  
import app.NGon;  
  
import javafx.scene.canvas.GraphicsContext;  
  
public final class NGonDrawer implements Drawer {  
   
 private static NGonDrawer *drawer* = null;  
 private final String className = NGon.class.getName();  
  
 @Override  
 public String getClassName() {  
 return className;  
 }  
  
 @Override  
 public void draw(IShape shape, GraphicsContext gc) throws Exception {  
 NGon ngon = (NGon) shape;  
 int n = ngon.getN();  
 double[] xcoords = new double[n];  
 double[] ycoords = new double[n];  
 for (int i = 0; i < n; i++) {  
 xcoords[i] = ngon.getP()[i].getX(0) + *centerX*;  
 ycoords[i] = -ngon.getP()[i].getX(1) + *centerY*;  
 }  
  
 gc.strokePolygon(xcoords, ycoords, n);  
 }  
  
 public static NGonDrawer createInstance() {  
 if (*drawer* == null)  
 *drawer* = new NGonDrawer();  
   
 return *drawer*;  
 }  
}

PolylineDrawer.java:

package ui.drawers;  
  
import app.IShape;  
import app.Polyline;  
  
import javafx.scene.canvas.GraphicsContext;  
  
public final class PolylineDrawer implements Drawer {  
   
 private static PolylineDrawer *drawer* = null;  
 private final String className = Polyline.class.getName();  
  
 @Override  
 public String getClassName() {  
 return className;  
 }  
  
 @Override  
 public void draw(IShape shape, GraphicsContext gc) throws Exception {  
 Polyline polyline = (Polyline) shape;  
 int n = polyline.getN();  
 for (int i = 1; i < n; i++) {  
 gc.strokeLine(polyline.getP()[i - 1].getX(0) + *centerX*, -polyline.getP()[i - 1].getX(1) + *centerY*,  
 polyline.getP()[i].getX(0) + *centerX*, -polyline.getP()[i].getX(1) + *centerY*);  
 }  
 }  
  
 public static PolylineDrawer createInstance() {  
 if (*drawer* == null)  
 *drawer* = new PolylineDrawer();  
   
 return *drawer*;  
 }  
}

QGonDrawer.java:

package ui.drawers;  
  
import app.IShape;  
import app.QGon;  
  
import javafx.scene.canvas.GraphicsContext;  
  
public final class QGonDrawer implements Drawer {  
   
 private static QGonDrawer *drawer* = null;  
 private final String className = QGon.class.getName();  
  
 @Override  
 public String getClassName() {  
 return className;  
 }  
  
 @Override  
 public void draw(IShape shape, GraphicsContext gc) throws Exception {  
 QGon qgon = (QGon) shape;  
 int n = qgon.getN();  
 double[] xcoords = new double[n];  
 double[] ycoords = new double[n];  
 for (int i = 0; i < n; i++) {  
 xcoords[i] = qgon.getP()[i].getX(0) + *centerX*;  
 ycoords[i] = -qgon.getP()[i].getX(1) + *centerY*;  
 }  
  
 gc.strokePolygon(xcoords, ycoords, n);  
 }  
  
 public static QGonDrawer createInstance() {  
 if (*drawer* == null)  
 *drawer* = new QGonDrawer();  
   
 return *drawer*;  
 }  
}

RectangleDrawer.java:

package ui.drawers;  
  
import app.IShape;  
import app.Rectangle;  
  
import javafx.scene.canvas.GraphicsContext;  
  
public final class RectangleDrawer implements Drawer {  
   
 private static RectangleDrawer *drawer* = null;  
 private final String className = Rectangle.class.getName();  
  
 @Override  
 public String getClassName() {  
 return className;  
 }  
  
 @Override  
 public void draw(IShape shape, GraphicsContext gc) throws Exception {  
 Rectangle rectangle = (Rectangle) shape;  
 int n = rectangle.getN();  
 double[] xcoords = new double[n];  
 double[] ycoords = new double[n];  
 for (int i = 0; i < n; i++) {  
 xcoords[i] = rectangle.getP()[i].getX(0) + *centerX*;  
 ycoords[i] = -rectangle.getP()[i].getX(1) + *centerY*;  
 }  
  
 gc.strokePolygon(xcoords, ycoords, n);  
 }  
  
 public static RectangleDrawer createInstance() {  
 if (*drawer* == null)  
 *drawer* = new RectangleDrawer();  
   
 return *drawer*;  
 }  
}

SegmentDrawer.java:

package ui.drawers;  
  
import app.IShape;  
import app.Segment;  
  
import javafx.scene.canvas.GraphicsContext;  
  
public final class SegmentDrawer implements Drawer {  
   
 private static SegmentDrawer *drawer* = null;  
 private final String className = Segment.class.getName();  
  
 @Override  
 public String getClassName() {  
 return className;  
 }  
  
 @Override  
 public void draw(IShape shape, GraphicsContext gc) throws Exception {  
 Segment segment = (Segment) shape;  
 gc.strokeLine(segment.getStart().getX(0) + *centerX*, -segment.getStart().getX(1) + *centerY*,  
 segment.getFinish().getX(0) + *centerX*, -segment.getFinish().getX(1) + *centerY*);  
 }  
  
 public static SegmentDrawer createInstance() {  
 if (*drawer* == null)  
 *drawer* = new SegmentDrawer();  
   
 return *drawer*;  
 }  
}

TGonDrawer.java:

package ui.drawers;  
  
import app.IShape;  
import app.TGon;  
  
import javafx.scene.canvas.GraphicsContext;  
  
public final class TGonDrawer implements Drawer {  
   
 private static TGonDrawer *drawer* = null;  
 private final String className = TGon.class.getName();  
  
 @Override  
 public String getClassName() {  
 return className;  
 }  
  
 @Override  
 public void draw(IShape shape, GraphicsContext gc) throws Exception {  
 TGon tgon = (TGon) shape;  
 int n = tgon.getN();  
 double[] xcoords = new double[n];  
 double[] ycoords = new double[n];  
 for (int i = 0; i < n; i++) {  
 xcoords[i] = tgon.getP()[i].getX(0) + *centerX*;  
 ycoords[i] = -tgon.getP()[i].getX(1) + *centerY*;  
 }  
  
 gc.strokePolygon(xcoords, ycoords, n);  
 }  
  
 public static TGonDrawer createInstance() {  
 if (*drawer* == null)  
 *drawer* = new TGonDrawer();  
   
 return *drawer*;  
 }  
}

TrapezeDrawer.java:

package ui.drawers;  
  
import app.IShape;  
import app.Trapeze;  
  
import javafx.scene.canvas.GraphicsContext;  
  
public final class TrapezeDrawer implements Drawer {  
  
 private static TrapezeDrawer *drawer* = null;  
 private final String className = Trapeze.class.getName();  
  
 @Override  
 public String getClassName() {  
 return className;  
 }  
  
 @Override  
 public void draw(IShape shape, GraphicsContext gc) throws Exception {  
 Trapeze trapeze = (Trapeze) shape;  
 int n = trapeze.getN();  
 double[] xcoords = new double[n];  
 double[] ycoords = new double[n];  
 for (int i = 0; i < n; i++) {  
 xcoords[i] = trapeze.getP()[i].getX(0) + *centerX*;  
 ycoords[i] = -trapeze.getP()[i].getX(1) + *centerY*;  
 }  
  
 gc.strokePolygon(xcoords, ycoords, n);  
 }  
  
 public static TrapezeDrawer createInstance() {  
 if (*drawer* == null)  
 *drawer* = new TrapezeDrawer();  
   
 return *drawer*;  
 }  
}

MongoDb.java:

package ui.repository;  
  
import org.bson.Document;  
  
import com.mongodb.client.MongoClient;  
import com.mongodb.client.MongoCollection;  
import com.mongodb.client.MongoDatabase;  
  
public final class MongoDb {  
   
 private static final String *geometryDBname* = "Geometry";  
 private static final String *geometryCollectionName* = "Figures";  
  
 private static MongoDb *inst* = null;  
 private String dbname = null;  
 private MongoDatabase database = null;  
  
 private MongoDb(MongoClient client, String dbname) throws IllegalStateException {  
 this.dbname = dbname;  
 database = client.getDatabase(dbname);  
 }  
  
 public static MongoDb create(MongoClient client, String dbname) throws IllegalStateException {  
 if (dbname == null)  
 dbname = *geometryDBname*;  
  
 if (*inst* == null) {  
 Object obj = new Object();  
 synchronized(obj) {  
 if (*inst* == null)  
 *inst* = new MongoDb(client, dbname);  
 }  
 }  
  
 return *inst*;  
 }  
  
 public static void clear() {  
 *inst*.database = null;  
 *inst* = null;  
 }  
  
 public MongoCollection<Document> getCollection(String collectionName) {  
 if (collectionName == null)  
 collectionName = *geometryCollectionName*;  
  
 MongoCollection<Document> collection = null;  
  
 try {  
 collection = database.getCollection(collectionName);  
 } catch (IllegalStateException e) {  
 collection = database.getCollection(*geometryCollectionName*);  
 e.printStackTrace();  
 }  
  
 return collection;  
 }  
   
}

Папка test в папке data-module:

CircleTest.java:

import static org.junit.Assert.*assertFalse*;  
import static org.junit.Assert.*assertTrue*;  
  
import org.junit.Test;  
  
import app.Circle;  
import app.Point2D;  
  
public class CircleTest {  
 private Point2D center;  
 private double radius = 2;  
 private Circle circle;  
  
 public CircleTest() {  
 try {  
 center = new Point2D(1, 2);  
 circle = new Circle(center, radius);  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
  
 @Test  
 public void checkSquare() {  
 String testName = "circle square";  
 try {  
 *assertTrue*(testName, circle.square() == Math.*PI* \* radius \* radius);  
 } catch (Exception e) {  
 e.printStackTrace();  
 *assertTrue*(testName, false);  
 }  
 }  
  
 @Test  
 public void checkLength() {  
 String testName = "circle length";  
 try {  
 *assertTrue*(testName, circle.length() == Math.*PI* \* 2 \* radius);  
 } catch (Exception e) {  
 e.printStackTrace();  
 *assertTrue*(testName, false);  
 }  
 }  
  
 @Test  
 public void checkShift() {  
 String testName = "circle shift";  
 try {  
 Point2D shiftVec = new Point2D(1, -2);  
 circle.shift(shiftVec);  
 *assertTrue*(testName, circle.getP().getX(0) == 2 && circle.getP().getX(1) == 0);  
 } catch (Exception e) {  
 e.printStackTrace();  
 *assertTrue*(testName, false);  
 }  
 }  
  
 @Test  
 public void checkCross() {  
 String testName = "circle cross";  
 try {  
 Point2D center1 = new Point2D(1, -2);  
 double radius1 = 2;  
 Circle circle1 = new Circle(center1, radius1);  
 *assertTrue*(testName, circle.cross(circle1));  
   
 radius1 = 1;  
 circle1.setR(radius1);  
 *assertFalse*(testName, circle.cross(circle1));  
 } catch (Exception e) {  
 e.printStackTrace();  
 *assertTrue*(testName, false);  
 }  
 }  
}

NGonTest.java:

import static org.junit.Assert.*assertTrue*;  
  
import org.junit.Test;  
  
import app.NGon;  
import app.Point2D;  
  
public class NGonTest {  
 private Point2D[] points;  
 private NGon ngon;  
  
 public NGonTest() {  
 try {  
 points = new Point2D[] {  
 new Point2D(1, 1),  
 new Point2D(2, 1),  
 new Point2D(3, 0),  
 new Point2D(4, -2),  
 new Point2D(0, 0)  
 };  
 ngon = new NGon(points);  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
  
 @Test  
 public void checkLength() {  
 String testName = "ngon length";  
 try {  
 *assertTrue*(testName, Math.*abs*(ngon.length() - 10.54) < 0.01);  
 } catch (Exception e) {  
 e.printStackTrace();  
 *assertTrue*(testName, false);  
 }  
 }  
  
 @Test  
 public void checkSquare() {  
 String testName = "ngon square";  
 try {  
 *assertTrue*(testName, Math.*abs*(ngon.square() - 5) < 0.01);  
 } catch (Exception e) {  
 e.printStackTrace();  
 *assertTrue*(testName, false);  
 }  
 }  
  
 @Test  
 public void checkCross() {  
 String testName = "ngon shift";  
 try {  
 Point2D[] points1 = new Point2D[] {  
 new Point2D(-2, 1),  
 new Point2D(-3, 1),  
 new Point2D(-4, 0),  
 new Point2D(-5, -2),  
 new Point2D(-1, 0)  
 };  
 NGon ngon1 = new NGon(points1);  
 // assertTrue(testName, ngon.cross(ngon1));  
  
 double rotAngle = -Math.*PI*;  
 ngon1.rot(rotAngle);  
 *assertTrue*(testName, ngon.cross(ngon1));  
 } catch (Exception e) {  
 e.printStackTrace();  
 *assertTrue*(testName, false);  
 }  
 }  
}

Point2DTest.java:

import static org.junit.Assert.*assertTrue*;  
  
import org.junit.Test;  
  
import app.Point2D;  
  
public class Point2DTest {  
 private Point2D point;  
  
 public Point2DTest() {  
 try {  
 point = new Point2D(new double[] {1, 3});  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
  
 @Test  
 public void checkRot() {  
 String testName = "point2D length";  
 *assertTrue*(testName, Math.*abs*(point.abs() - 3.1623) < 0.001);  
  
 try {  
 point = Point2D.*rot*(point, Math.*PI*);  
 *assertTrue*(testName, Math.*abs*(Point2D.*add*(point, point).abs() - 6.3246) < 0.001);  
 } catch (Exception e) {  
 e.printStackTrace();  
 *assertTrue*(testName, false);  
 }  
 }  
}

PointTest.java:

import static org.junit.Assert.*assertTrue*;  
  
import org.junit.Test;  
  
import app.Point;  
  
public class PointTest {  
   
 private Point point;  
  
 public PointTest() {  
 try {  
 point = new Point(5, new double[] {1, 2, 3, 4, 5});  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
  
 @Test  
 public void checkLength() {  
 String testName = "point length";  
 *assertTrue*(testName, Math.*abs*(point.abs() - 7.4162) < 0.001);  
 }  
  
 @Test  
 public void checkAdd() {  
 String testName = "point add";  
 try {  
 *assertTrue*(testName, Math.*abs*(Point.*add*(point, point).abs() - 14.8324) < 0.001);  
 } catch (Exception e) {  
 e.printStackTrace();  
 *assertTrue*(testName, false);  
 }  
 }  
  
 @Test  
 public void checkSub() {  
 String testName = "point sub";  
 try {  
 *assertTrue*(testName, Math.*abs*(Point.*sub*(point, point).abs()) < 0.01);  
 } catch (Exception e) {  
 e.printStackTrace();  
 *assertTrue*(testName, false);  
 }  
 }  
  
 @Test  
 public void checkMulti() {  
 String testName = "point multi";  
 try {  
 *assertTrue*(testName, Math.*abs*(Point.*mult*(point, 3).abs() - 22.2486) < 0.001);  
 } catch (Exception e) {  
 e.printStackTrace();  
 *assertTrue*(testName, false);  
 }  
 }  
}

PolylineTest.java:

import static org.junit.Assert.*assertFalse*;  
import static org.junit.Assert.*assertTrue*;  
  
import org.junit.Test;  
  
import app.Point2D;  
import app.Polyline;  
  
public class PolylineTest {  
   
 private Polyline polyline;  
 private Point2D[] point;  
  
 public PolylineTest() {  
 try {  
 point = new Point2D[] {  
 new Point2D(1, 3),  
 new Point2D(2, 4)  
 };  
 polyline = new Polyline(point);  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
  
 @Test  
 public void checkLength() {  
 String testName = "Polyline length";  
 try {  
 *assertTrue*(testName, Math.*abs*(polyline.length() - 1.4142) < 0.001);  
 } catch (Exception e) {  
 e.printStackTrace();  
 *assertTrue*(testName, false);  
 }  
 }  
  
 @Test  
 public void checkCross() {  
 String testName = "Polyline cross";  
 try {  
 Point2D[] points = new Point2D[] {  
 new Point2D(),  
 new Point2D(2, 3),  
 new Point2D(1, 4)  
 };  
 Polyline polyline1 = new Polyline(points);  
 *assertTrue*(testName, polyline1.cross(polyline));  
  
 Point2D shiftVec = new Point2D(0, -2);  
 polyline1.shift(shiftVec);  
 *assertFalse*(testName, polyline1.cross(polyline));  
 } catch (Exception e) {  
 e.printStackTrace();  
 *assertTrue*(testName, false);  
 }  
 }  
}

QGonTest.java:

import static org.junit.Assert.*assertTrue*;  
  
import org.junit.Test;  
  
import app.Point2D;  
import app.QGon;  
  
public class QGonTest {  
   
 private Point2D[] points;  
 private QGon qgon;  
  
 public QGonTest() {  
 try {  
 points = new Point2D[] {  
 new Point2D(),  
 new Point2D(1, 1),  
 new Point2D(5, 1),  
 new Point2D(5, -1)  
 };  
 qgon = new QGon(points);  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
   
 @Test  
 public void checkLength() {  
 String testName = "qgon length";  
  
 try {  
 *assertTrue*(testName, Math.*abs*(qgon.length() - 12.51) < 0.01);  
 } catch (Exception e) {  
 e.printStackTrace();  
 *assertTrue*(testName, false);  
 }  
 }  
  
 @Test  
 public void checkSquare() {  
 String testName = "qgon square";  
  
 try {  
 *assertTrue*(testName, Math.*abs*(qgon.square() - 7.00) < 0.01);  
 } catch (Exception e) {  
 e.printStackTrace();  
 *assertTrue*(testName, false);  
 }  
 }  
}

RectangleTest.java:

import static org.junit.Assert.*assertTrue*;  
  
import org.junit.Test;  
  
import app.Point2D;  
import app.Rectangle;  
  
public class RectangleTest {  
   
 private Rectangle rectangle;  
 private Point2D[] points;  
  
 public RectangleTest() {  
 try {  
 points = new Point2D[] {  
 new Point2D(),  
 new Point2D(0, 10),  
 new Point2D(10, 10),  
 new Point2D(10, 0)  
 };  
 rectangle = new Rectangle(points);  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
  
 @Test  
 public void checkLength() {  
 String testName = "rectangle length";  
 try {  
 *assertTrue*(testName, Math.*abs*(rectangle.length() - 40.00) < 0.01);  
 } catch (Exception e) {  
 e.printStackTrace();  
 *assertTrue*(testName, false);  
 }  
 }  
  
 @Test  
 public void checkSquare() {  
 String testName = "rectangle square";  
 try {  
 *assertTrue*(testName, Math.*abs*(rectangle.square() - 100.00) < 0.01);  
 } catch (Exception e) {  
 e.printStackTrace();  
 *assertTrue*(testName, false);  
 }  
 }  
}

SegmentTest.java:

import static org.junit.Assert.*assertTrue*;  
  
import org.junit.Test;  
  
import app.Point2D;  
import app.Segment;  
  
public class SegmentTest {  
   
 private Segment segment;  
 private Point2D[] points;  
  
 public SegmentTest() {  
 try {  
 points = new Point2D[] {  
 new Point2D(),  
 new Point2D(5, 10),  
 };  
 segment = new Segment(points[0], points[1]);  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
  
 @Test  
 public void checkLength() {  
 String testName = "segment length";  
 try {  
 *assertTrue*(testName, Math.*abs*(segment.length() - 11.1803) < 0.001);  
 } catch (Exception e) {  
 e.printStackTrace();  
 *assertTrue*(testName, false);  
 }  
 }  
  
 @Test  
 public void checkSquare() {  
 String testName = "segment square";  
 try {  
 *assertTrue*(testName, Math.*abs*(segment.square()) < 0.01);  
 } catch (Exception e) {  
 e.printStackTrace();  
 *assertTrue*(testName, false);  
 }  
 }  
}

TGonTest.java:

import static org.junit.Assert.*assertTrue*;  
  
import org.junit.Test;  
  
import app.Point2D;  
import app.TGon;  
  
public class TGonTest {  
  
 private TGon tgon;  
 private Point2D[] points;  
  
 public TGonTest() {  
 try {  
 points = new Point2D[] {  
 new Point2D(),  
 new Point2D(5, 10),  
 new Point2D(10, 0)  
 };  
 tgon = new TGon(points);  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
  
 @Test  
 public void checkLength() {  
 String testName = "tgon length";  
 try {  
 *assertTrue*(testName, Math.*abs*(tgon.length() - 32.3607) < 0.001);  
 } catch (Exception e) {  
 e.printStackTrace();  
 *assertTrue*(testName, false);  
 }  
 }  
  
 @Test  
 public void checkSquare() {  
 String testName = "tgon square";  
 try {  
 *assertTrue*(testName, Math.*abs*(tgon.square() - 50.00) < 0.01);  
 } catch (Exception e) {  
 e.printStackTrace();  
 *assertTrue*(testName, false);  
 }  
 }  
}

TrapezeTest.java:

import static org.junit.Assert.*assertTrue*;  
  
import org.junit.Test;  
  
import app.Point2D;  
import app.Trapeze;  
  
public class TrapezeTest {  
   
 private Point2D[] points;  
 private Trapeze trapeze;  
  
 public TrapezeTest() {  
 try {  
 points = new Point2D[] {  
 new Point2D(),  
 new Point2D(4, 6),  
 new Point2D(11, 6),  
 new Point2D(12, 0)  
 };  
 trapeze = new Trapeze(points);  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
  
 @Test  
 public void checkLength() {  
 String testName = "trapeze length";  
 try {  
 *assertTrue*(testName, Math.*abs*(trapeze.length() - 32.2939) < 0.001);  
 } catch (Exception e) {  
 e.printStackTrace();  
 *assertTrue*(testName, false);  
 }  
 }   
}

build.gradle:

plugins **{** id 'java-library'  
 id 'java'  
 id "io.franzbecker.gradle-lombok" version '5.0.0'  
**}**lombok **{** version = '1.18.4'  
 sha256 = ""  
**}**dependencies **{  
}**

## Android-приложение

AndroidManifest.xml:

<?xml version="1.0" encoding="utf-8"?>  
<manifest xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:tools="http://schemas.android.com/tools"  
 package="com.example.geometry">  
  
 <uses-permission android:name="android.permission.INTERNET" />  
 <uses-permission android:name="android.permission.ACCESS\_NETWORK\_STATE" />  
 <uses-permission android:name="android.permission.READ\_EXTERNAL\_STORAGE" />  
 <uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE" />  
 <uses-permission android:name="android.permission.MANAGE\_EXTERNAL\_STORAGE" />  
  
 <application  
 android:name=".utils.GeometryApplication"  
 android:allowBackup="true"  
 android:dataExtractionRules="@xml/data\_extraction\_rules"  
 android:fullBackupContent="@xml/backup\_rules"  
 android:icon="@mipmap/ic\_launcher"  
 android:label="@string/app\_name"  
 android:roundIcon="@mipmap/ic\_launcher\_round"  
 android:supportsRtl="true"  
 android:usesCleartextTraffic="true"  
 android:theme="@style/Theme.JavaGeometryExample"  
 tools:targetApi="31">  
 <activity  
 android:name=".views.MainActivity"  
 android:exported="true">  
 <intent-filter>  
 <action android:name="android.intent.action.MAIN" />  
  
 <category android:name="android.intent.category.LAUNCHER" />  
 </intent-filter>  
 </activity>  
 </application>  
  
</manifest>

AddFigureData.java:

package com.example.geometry.callbacks;  
  
import com.example.geometry.utils.CallbackData;  
  
import java.util.List;  
  
public class AddFigureData implements CallbackData {  
 private String type;  
 private List<Double> coords;  
  
 public AddFigureData(String type, List<Double> coords) {  
 this.type = type;  
 this.coords = coords;  
 }  
  
 public String getType() {  
 return type;  
 }  
  
 public List<Double> getCoords() {  
 return coords;  
 }  
}

CheckIfCrossData.java:

package com.example.geometry.callbacks;  
  
import com.example.geometry.utils.CallbackData;  
  
public class CheckIfCrossData implements CallbackData {  
  
 private int first;  
 private int second;  
  
 public CheckIfCrossData(int first, int second) {  
 this.first = first;  
 this.second = second;  
 }  
  
 public int getFirst() {  
 return first;  
 }  
  
 public int getSecond() {  
 return second;  
 }  
}

CountData.java:

package com.example.geometry.callbacks;  
  
import com.example.geometry.utils.CallbackData;  
  
public class CountData implements CallbackData {  
  
 private int position;  
 private boolean type;  
  
 public CountData(int position, boolean type) {  
 this.position = position;  
 this.type = type;  
 }  
  
 public int getFirst() {  
 return position;  
 }  
  
 public boolean getType() {  
 return type;  
 }  
}

DragFigureData.java:

package com.example.geometry.callbacks;  
  
import com.example.geometry.utils.CallbackData;  
  
import java.util.List;  
  
public class DragFigureData implements CallbackData {  
 private final int index;  
 private final String type;  
 private final List<Double> coords;  
  
 public DragFigureData(int index, String type, List<Double> coords) {  
 this.index = index;  
 this.type = type;  
 this.coords = coords;  
 }  
  
 public int getIndex() {  
 return index;  
 }  
  
 public String getType() {  
 return type;  
 }  
  
 public List<Double> getCoords() {  
 return coords;  
 }  
}

RemoveData.java:

package com.example.geometry.callbacks;  
  
import com.example.geometry.utils.CallbackData;  
  
public class RemoveData implements CallbackData {  
  
 private int position;  
  
 public RemoveData(int position) {  
 this.position = position;  
 }  
  
 public int getPosition() {  
 return position;  
 }  
}

GeometryApplication.java:

package com.example.geometry.utils;  
  
import android.app.Application;  
  
import com.example.common\_utils.repository.MongoDb;  
  
public class GeometryApplication extends Application {  
  
 @Override  
 public void onCreate() {  
 super.onCreate();  
  
 MongoDb.*connectDB*(this);  
 }  
}

GeometryCanvas.java:

package com.example.geometry.utils;  
  
import android.content.Context;  
import android.graphics.Canvas;  
import android.graphics.Color;  
import android.graphics.Paint;  
import android.os.Parcel;  
import android.os.Parcelable;  
import android.util.AttributeSet;  
import android.util.Log;  
import android.view.View;  
  
import androidx.annotation.NonNull;  
  
import com.example.common\_utils.deserializers.IFigureFactory;  
import com.example.common\_utils.drawers.Drawer;  
import com.example.common\_utils.drawers.DrawerFactory;  
  
import java.util.ArrayList;  
import java.util.LinkedList;  
import java.util.List;  
import java.util.Queue;  
  
import app.IShape;  
  
public class GeometryCanvas extends View {  
  
 private ArrayList<IShape> list;  
 private ArrayList<IShape> redList;  
 private Paint paint;  
  
 private void init() {  
 list = new ArrayList<>();  
 redList = new ArrayList<>();  
 paint = new Paint();  
 paint.setColor(Color.*GRAY*);  
 }  
  
 public GeometryCanvas(Context context) {  
 super(context);  
 init();  
 }  
  
 public GeometryCanvas(Context context, AttributeSet attributeSet) {  
 super(context, attributeSet);  
 init();  
 }  
  
 public GeometryCanvas(Context context, AttributeSet attributeSet, int defStyleAttr) {  
 super(context, attributeSet, defStyleAttr);  
 init();  
 }  
  
 @Override  
 public Parcelable onSaveInstanceState() {  
 Parcelable superState = super.onSaveInstanceState();  
 Log.*e*("kek", Integer.*toString*(list.size()));  
 return new GeometryCanvasData(superState, list, redList);  
 }  
  
 @Override  
 public void onRestoreInstanceState(Parcelable p) {  
 GeometryCanvasData data = (GeometryCanvasData) p;  
 super.onRestoreInstanceState(data.getSuperState());  
  
 list = data.getShapes();  
 redList = data.getRedShapes();  
  
 while (!OperationStorage.*eventQueue*.isEmpty()) {  
 OperationStorage.Operation<?> operation = OperationStorage.*eventQueue*.poll();  
 assert operation != null;  
 if (operation.type == OperationStorage.Operation.OperationType.*ADD*) {  
 mAddToListAndDraw((IShape)operation.shape);  
 } else if (operation.type == OperationStorage.Operation.OperationType.*REMOVE*) {  
 mRemoveAll();  
 } else if (operation.type == OperationStorage.Operation.OperationType.*CLEAR*) {  
 mClearAndDrawAll((List<IShape>)operation.shape);  
 } else if (operation.type == OperationStorage.Operation.OperationType.*REMOVE\_ONE*) {  
 mRemove(operation.position);  
 }  
 }  
  
 Log.*e*("lol", Integer.*toString*(list.size()));  
 }  
  
 public List<IShape> getList() {  
 return list;  
 }  
  
 public void clearAndDrawAll(List<IShape> redList, boolean redrawNow) {  
 if (redrawNow) {  
 mClearAndDrawAll(redList);  
 invalidate();  
 } else {  
 OperationStorage.*eventQueue*.add(new OperationStorage.Operation<>(  
 OperationStorage.Operation.OperationType.*CLEAR*,  
 redList  
 ));  
 }  
 }  
  
 public void removeAll(boolean redrawNow) {  
 if (redrawNow) {  
 mRemoveAll();  
 invalidate();  
 } else {  
 OperationStorage.*eventQueue*.add(new OperationStorage.Operation<>(  
 OperationStorage.Operation.OperationType.*REMOVE*,  
 null  
 ));  
 }  
 }  
  
 public void addToListAndDraw(IShape shape, boolean redrawNow) {  
 if (redrawNow) {  
 mAddToListAndDraw(shape);  
 invalidate();  
 } else {  
 OperationStorage.*eventQueue*.add(new OperationStorage.Operation<>(  
 OperationStorage.Operation.OperationType.*ADD*,  
 shape  
 ));  
 }  
 }  
  
 public void remove(int position) {  
 OperationStorage.*eventQueue*.add(new OperationStorage.Operation<>(  
 OperationStorage.Operation.OperationType.*REMOVE\_ONE*,  
 position  
 ));  
 }  
  
 private void mClearAndDrawAll(List<IShape> redList) {  
 this.redList.clear();  
 if (redList != null)  
 this.redList.addAll(redList);  
 }  
  
 private void mRemoveAll() {  
 redList.clear();  
 list.clear();  
 }  
  
 private void mAddToListAndDraw(IShape shape) {  
 list.add(shape);  
 Log.*e*("tut", Integer.*toString*(list.size()));  
 }  
  
 private void mUpdate(int pos, IShape shape) {  
 list.remove(pos);  
 list.add(shape);  
 }  
  
 private void mRemove(int position) {  
 list.remove(position);  
 }  
  
 private boolean findIn(IShape shape, List<IShape> shapeList) {  
 if (shapeList == null || shape == null)  
 return false;  
  
 for (IShape curShape : shapeList) {  
 if (shape.toString().equals(curShape.toString()))  
 return true;  
 }  
  
 return false;  
 }  
  
 private void drawAxes(@NonNull Canvas canvas, Paint paint) throws Exception {  
 canvas.drawLine(  
 0,  
 (float) canvas.getHeight() / 2,  
 (float) canvas.getWidth(),  
 (float) canvas.getHeight() / 2,  
 paint  
 );  
 canvas.drawLine(  
 (float) canvas.getWidth() / 2,  
 (float) canvas.getHeight(),  
 (float) canvas.getWidth() / 2,  
 0,  
 paint  
 );  
 }  
  
 @Override  
 protected void onLayout(boolean changed, int left, int top, int right, int bottom) {  
 super.onLayout(changed, left, top, right, bottom);  
  
 Drawer.*setCenter*((double)getMeasuredWidth() / 2, (double)getMeasuredHeight() / 2);  
 }  
  
 @Override  
 protected void onDraw(Canvas canvas) {  
 Log.*e*(getClass().toString(), Integer.*toString*(list.size()));  
 try {  
 drawAxes(canvas, paint);  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
  
 list.forEach((shape) -> {  
 try {  
 if (findIn(shape, redList)) {  
 paint.setColor(Color.*RED*);  
 DrawerFactory.*create*(shape).draw(shape, canvas, paint);  
 paint.setColor(Color.*GRAY*);  
 } else {  
 DrawerFactory.*create*(shape).draw(shape, canvas, paint);  
 }  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 });  
 }  
  
 static final class GeometryCanvasData extends View.BaseSavedState {  
  
 private final ArrayList<String> shapes;  
 private final ArrayList<String> redShapes;  
  
 public GeometryCanvasData(Parcelable superState, ArrayList<IShape> shapes, ArrayList<IShape> redShapes) {  
 super(superState);  
 this.shapes = new ArrayList<>();  
 this.redShapes = new ArrayList<>();  
 shapes.forEach((shape) -> this.shapes.add(shape.toString()));  
 redShapes.forEach((shape) -> this.redShapes.add(shape.toString()));  
 }  
  
 public GeometryCanvasData(Parcel in) {  
 super(in);  
 shapes = new ArrayList<>();  
 redShapes = new ArrayList<>();  
 in.readStringList(shapes);  
 in.readStringList(redShapes);  
 }  
  
 public ArrayList<IShape> getShapes() {  
 ArrayList<IShape> resultShapes = new ArrayList<>();  
 shapes.forEach((shape) -> {  
 resultShapes.add(IFigureFactory.*create*(shape));  
 });  
  
 return resultShapes;  
 }  
  
 public ArrayList<IShape> getRedShapes() {  
 ArrayList<IShape> resultShapes = new ArrayList<>();  
 redShapes.forEach((shape) -> {  
 resultShapes.add(IFigureFactory.*create*(shape));  
 });  
  
 return resultShapes;  
 }  
  
 public static final Creator<GeometryCanvasData> *CREATOR* = new Creator<>() {  
 @Override  
 public GeometryCanvasData createFromParcel(Parcel in) {  
 return new GeometryCanvasData(in);  
 }  
  
 @Override  
 public GeometryCanvasData[] newArray(int size) {  
 return new GeometryCanvasData[size];  
 }  
 };  
  
 @Override  
 public int describeContents() {  
 return 0;  
 }  
  
 @Override  
 public void writeToParcel(Parcel parcel, int i) {  
 super.writeToParcel(parcel, i);  
 parcel.writeStringList(shapes);  
 parcel.writeStringList(redShapes);  
 }  
 }  
  
 private static final class OperationStorage {  
 public static Queue<Operation<?>> *eventQueue* = new LinkedList<>();  
  
 public static final class Operation<T> {  
 private OperationType type;  
 private T shape;  
 private int position = -1;  
  
 public Operation(OperationType type, T shape) {  
 this.type = type;  
 this.shape = shape;  
 }  
  
 public Operation(OperationType type, int position) {  
 this(type, null);  
 this.position = position;  
 }  
  
 public enum OperationType {  
 *REMOVE*,  
 *ADD*,  
 *CLEAR*,  
 *REMOVE\_ONE*,  
 }  
 }  
 }  
}

ICallbackHandler.java:

package com.example.geometry.utils;  
  
public interface ICallbackHandler<T extends CallbackData> {  
 void handleEvent(T data);  
}

Result.java:

package com.example.geometry.utils;  
  
public class Result<T> {  
 private boolean error;  
 private T data = null;  
  
 public Result(boolean error, T data) {  
 this.error = error;  
 this.data = data;  
 }  
  
 public boolean getError() {  
 return error;  
 }  
  
 public T getData() {  
 return data;  
 }  
}

AddFigureFragment.java:

package com.example.geometry.views;  
  
import android.os.Bundle;  
import android.view.LayoutInflater;  
import android.view.View;  
import android.view.ViewGroup;  
import android.widget.AdapterView;  
import android.widget.ArrayAdapter;  
import android.widget.Button;  
import android.widget.EditText;  
import android.widget.LinearLayout;  
import android.widget.Spinner;  
  
import androidx.annotation.NonNull;  
import androidx.annotation.Nullable;  
import androidx.fragment.app.Fragment;  
  
import com.example.geometry.R;  
import com.example.geometry.callbacks.AddFigureData;  
import com.example.geometry.utils.ICallbackHandler;  
  
import java.util.ArrayList;  
import java.util.List;  
  
public class AddFigureFragment extends Fragment {  
  
 private Spinner spinner = null;  
 private Button addFigure = null;  
 private Button cancel = null;  
 private Spinner pointNumber = null;  
 private LinearLayout xLayout = null;  
 private LinearLayout yLayout = null;  
 private String selectedFigure = null;  
 private ICallbackHandler<AddFigureData> callback = null;  
  
 private AddFigureFragment(ICallbackHandler<AddFigureData> callback) {  
 this.callback = callback;  
 }  
  
 public static AddFigureFragment createInstance(ICallbackHandler<AddFigureData> callback) {  
 return new AddFigureFragment(callback);  
 }  
  
 @Override  
 @Nullable  
 public View onCreateView(@NonNull LayoutInflater inflater, @Nullable ViewGroup container,  
 @Nullable Bundle savedInstanceState) {  
 return inflater.inflate(R.layout.*fragment\_add\_figure*, container, false);  
 }  
  
 @Override  
 public void onViewCreated(@NonNull View view, @Nullable Bundle savedInstanceState) {  
 super.onViewCreated(view, savedInstanceState);  
  
 spinner = view.findViewById(R.id.*types*);  
  
 addFigure = view.findViewById(R.id.*add*);  
 addFigure.setOnClickListener(this::onAddFigure);  
  
 cancel = view.findViewById(R.id.*cancel*);  
 cancel.setOnClickListener(this::onCancel);  
  
 xLayout = view.findViewById(R.id.*xLayout*);  
 yLayout = view.findViewById(R.id.*yLayout*);  
  
 pointNumber = view.findViewById(R.id.*pointNumber*);  
  
 setPointNumber();  
 setDataToCombo();  
 }  
  
 @Override  
 public void onDestroy() {  
 spinner = null;  
 addFigure = null;  
 cancel = null;  
 xLayout = null;  
 yLayout = null;  
 pointNumber = null;  
 super.onDestroy();  
 }  
  
 public void setDataToCombo() {  
 ArrayAdapter<CharSequence> adapter = ArrayAdapter.*createFromResource*(  
 this.getContext(),  
 R.array.*shape\_types*,  
 android.R.layout.*simple\_list\_item\_1* );  
 adapter.setDropDownViewResource(android.R.layout.*simple\_spinner\_dropdown\_item*);  
 spinner.setAdapter(adapter);  
  
 spinner.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {  
 @Override  
 public void onItemSelected(AdapterView<?> adapterView, View view, int i, long l) {  
 String[] choose = getResources().getStringArray(R.array.*shape\_types*);  
 AddFigureFragment.this.addAttrs(choose[i]);  
 }  
  
 @Override  
 public void onNothingSelected(AdapterView<?> adapterView) {  
 // pass  
 }  
 });  
 }  
  
 public void addAttrs(String type) {  
 selectedFigure = type;  
 switch (type) {  
 case "Circle":  
 addCircle();  
 break;  
 case "Segment":  
 addSegment();  
 break;  
 case "TGon":  
 addTGon();  
 break;  
 case "NGon":  
 addNGon();  
 break;  
 case "QGon":  
 addQGon();  
 break;  
 case "Trapeze":  
 addTrapeze();  
 break;  
 case "Rectangle":  
 addRectangle();  
 break;  
 case "Polyline":  
 addPolyline();  
 break;  
 }  
 }  
  
 private void onAddFigure(View view) {  
 List<Double> arr = new ArrayList<>();  
 for (int pos = 0; pos < xLayout.getChildCount(); pos++) {  
 if (xLayout.getChildAt(pos).getVisibility() == View.*INVISIBLE*) {  
 break;  
 }  
 arr.add(Double.*parseDouble*(((EditText)xLayout.getChildAt(pos)).getText().toString()));  
 if (yLayout.getChildAt(pos).getVisibility() == View.*VISIBLE*) {  
 arr.add(Double.*parseDouble*(((EditText)yLayout.getChildAt(pos)).getText().toString()));  
 }  
 }  
  
 callback.handleEvent(new AddFigureData(selectedFigure, arr));  
 onCancel(view);  
 }  
  
 private void onCancel(View view) {  
 getParentFragmentManager().popBackStack();  
 }  
  
 private void removePointNumber() {  
 if (pointNumber != null) {  
 pointNumber.setVisibility(View.*INVISIBLE*);  
 }  
 }  
  
 private void addPointNumber() {  
 if (pointNumber != null) {  
 pointNumber.setVisibility(View.*VISIBLE*);  
 }  
 }  
  
 private void setPointNumber() {  
 ArrayAdapter<CharSequence> adapter = ArrayAdapter.*createFromResource*(  
 this.getContext(),  
 R.array.*number\_of\_vertices*,  
 android.R.layout.*simple\_list\_item\_1* );  
 adapter.setDropDownViewResource(android.R.layout.*simple\_spinner\_dropdown\_item*);  
 pointNumber.setAdapter(adapter);  
  
 pointNumber.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {  
 @Override  
 public void onItemSelected(AdapterView<?> adapterView, View view, int i, long l) {  
 String[] choose = getResources().getStringArray(R.array.*number\_of\_vertices*);  
 clearLayout();  
 addNFields(Integer.*parseInt*(choose[i]));  
 }  
  
 @Override  
 public void onNothingSelected(AdapterView<?> adapterView) {  
 // pass  
 }  
 });  
 }  
  
 private void clearLayout() {  
 for (int pos = 0; pos < xLayout.getChildCount(); pos++) {  
 View child = xLayout.getChildAt(pos);  
 child.setVisibility(View.*INVISIBLE*);  
  
 child = yLayout.getChildAt(pos);  
 child.setVisibility(View.*INVISIBLE*);  
 }  
 }  
  
 private void changeFieldVisibility(int position) {  
 View child = xLayout.getChildAt(position);  
 if (child.getVisibility() == View.*VISIBLE*) {  
 child.setVisibility(View.*INVISIBLE*);  
 } else {  
 child.setVisibility(View.*VISIBLE*);  
 }  
 }  
  
 private void addNFields(int n) {  
 for (int pos = 0; pos < Math.*min*(n, xLayout.getChildCount()); pos++) {  
 View xChild = xLayout.getChildAt(pos);  
 View yChild = yLayout.getChildAt(pos);  
 xChild.setVisibility(View.*VISIBLE*);  
 yChild.setVisibility(View.*VISIBLE*);  
 }  
 }  
  
 private void addCircle() {  
 removePointNumber();  
 clearLayout();  
 addNFields(1);  
 changeFieldVisibility(1);  
 }  
  
 private void addSegment() {  
 removePointNumber();  
 clearLayout();  
 addNFields(2);  
 }  
  
 private void addTGon() {  
 removePointNumber();  
 clearLayout();  
 addNFields(3);  
 }  
  
 private void addNGon() {  
 removePointNumber();  
 clearLayout();  
 addPointNumber();  
 String[] choose = getResources().getStringArray(R.array.*number\_of\_vertices*);  
 addNFields(Integer.*parseInt*(choose[pointNumber.getSelectedItemPosition()]));  
 }  
  
 private void addQGon() {  
 removePointNumber();  
 clearLayout();  
 addNFields(4);  
 }  
  
 private void addTrapeze() {  
 removePointNumber();  
 addQGon();  
 }  
  
 private void addRectangle() {  
 removePointNumber();  
 addQGon();  
 }  
  
 private void addPolyline() {  
 addNGon();  
 }  
  
}

CheckIfCrossFragment.java:

package com.example.geometry.views;  
  
import android.os.Bundle;  
import android.view.LayoutInflater;  
import android.view.View;  
import android.view.ViewGroup;  
import android.widget.AdapterView;  
import android.widget.ArrayAdapter;  
import android.widget.Button;  
import android.widget.Spinner;  
import android.widget.Toast;  
  
import androidx.annotation.NonNull;  
import androidx.annotation.Nullable;  
import androidx.fragment.app.Fragment;  
  
import com.example.geometry.R;  
import com.example.geometry.callbacks.CheckIfCrossData;  
import com.example.geometry.utils.ICallbackHandler;  
  
import java.util.List;  
import java.util.stream.Collectors;  
  
public class CheckIfCrossFragment extends Fragment {  
  
 private final ICallbackHandler<CheckIfCrossData> callback;  
 private final List<String> currentFigures;  
  
 private Spinner type = null;  
 private Spinner firstShape = null;  
 private Spinner secondShape = null;  
 private Button checkIfCross = null;  
 private Button cancel = null;  
  
 private CheckIfCrossFragment(ICallbackHandler<CheckIfCrossData> callback,  
 List<String> currentFigures) {  
 this.callback = callback;  
 this.currentFigures = currentFigures;  
 }  
  
 public static CheckIfCrossFragment createInstance(ICallbackHandler<CheckIfCrossData> callback,  
 List<String> currentFigures) {  
 return new CheckIfCrossFragment(callback, currentFigures);  
 }  
  
 @Override  
 @Nullable  
 public View onCreateView(@NonNull LayoutInflater inflater, @Nullable ViewGroup container,  
 @Nullable Bundle savedInstanceState) {  
 return inflater.inflate(R.layout.*fragment\_check\_cross*, container, false);  
 }  
  
 @Override  
 public void onViewCreated(@NonNull View view, @Nullable Bundle savedInstanceState) {  
 super.onViewCreated(view, savedInstanceState);  
  
 cancel = view.findViewById(R.id.*cancel*);  
 cancel.setOnClickListener(this::onCancel);  
  
 checkIfCross = view.findViewById(R.id.*checkIfCross*);  
 checkIfCross.setOnClickListener(this::onCheckIfCross);  
  
 firstShape = view.findViewById(R.id.*firstShape*);  
 secondShape = view.findViewById(R.id.*secondShape*);  
 type = view.findViewById(R.id.*type*);  
  
 setDataToCombo();  
 }  
  
 private void onCheckIfCross(View view) {  
 String firstOne = (String) firstShape.getSelectedItem();  
 String secondOne = (String) secondShape.getSelectedItem();  
  
 if (firstOne == null || secondOne == null  
 || firstOne.equals("") || secondOne.equals("")) {  
 showAlert("Data invalid");  
 return;  
 }  
  
 int firstPosition = -1;  
 int secondPosition = 0;  
 for (int i = 0; i < currentFigures.size(); i++) {  
 if (currentFigures.get(i).equals(firstOne)) {  
 firstPosition = i;  
 }  
  
 if (currentFigures.get(i).equals(secondOne)) {  
 secondPosition = i;  
 }  
 }  
  
 callback.handleEvent(new CheckIfCrossData(firstPosition, secondPosition));  
 onCancel(view);  
 }  
  
 private void onCancel(View view) {  
 getParentFragmentManager().popBackStack();  
 }  
  
 private void setDataToTypes(String shapeTypeFilter) {  
 List<String> filtered = currentFigures.stream()  
 .filter((shapeConstr) -> shapeConstr.split("\\(")[0].equals(shapeTypeFilter))  
 .collect(Collectors.*toList*());  
  
 ArrayAdapter<String> adapter = new ArrayAdapter<>(this.getContext(),  
 R.layout.*small\_spinner\_item*,  
 filtered);  
  
 adapter.setDropDownViewResource(android.R.layout.*simple\_spinner\_dropdown\_item*);  
 firstShape.setAdapter(adapter);  
 secondShape.setAdapter(adapter);  
 }  
  
 private void setDataToCombo() {  
 ArrayAdapter<CharSequence> adapter = ArrayAdapter.*createFromResource*(  
 this.getContext(),  
 R.array.*shape\_types*,  
 android.R.layout.*simple\_list\_item\_1* );  
 adapter.setDropDownViewResource(android.R.layout.*simple\_spinner\_dropdown\_item*);  
 type.setAdapter(adapter);  
  
 type.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {  
 @Override  
 public void onItemSelected(AdapterView<?> adapterView, View view, int i, long l) {  
 String[] choose = getResources().getStringArray(R.array.*shape\_types*);  
 CheckIfCrossFragment.this.setDataToTypes(choose[i]);  
 }  
  
 @Override  
 public void onNothingSelected(AdapterView<?> adapterView) {  
 // pass  
 }  
 });  
 }  
  
 private void showAlert(CharSequence text) {  
 Toast.*makeText*(getContext(),  
 text,  
 Toast.*LENGTH\_SHORT*).show();  
 }  
}

CountFragment.java:

package com.example.geometry.views;  
  
import android.os.Bundle;  
import android.view.LayoutInflater;  
import android.view.View;  
import android.view.ViewGroup;  
import android.widget.ArrayAdapter;  
import android.widget.Button;  
import android.widget.Spinner;  
  
import androidx.annotation.NonNull;  
import androidx.annotation.Nullable;  
import androidx.fragment.app.Fragment;  
  
import com.example.geometry.R;  
import com.example.geometry.callbacks.CountData;  
import com.example.geometry.utils.ICallbackHandler;  
  
import java.util.List;  
  
public class CountFragment extends Fragment {  
  
 private final ICallbackHandler<CountData> callback;  
 private final List<String> currentFigures;  
 private final boolean countType;  
  
 private Button cancel = null;  
 private Button count = null;  
 private Spinner type = null;  
  
 private CountFragment(ICallbackHandler<CountData> callback,  
 List<String> currentFigures, boolean type) {  
 this.callback = callback;  
 this.currentFigures = currentFigures;  
 this.countType = type;  
 }  
  
 public static CountFragment createInstance(ICallbackHandler<CountData> callback,  
 List<String> currentFigures, boolean type) {  
 return new CountFragment(callback, currentFigures, type);  
 }  
  
 @Override  
 @Nullable  
 public View onCreateView(@NonNull LayoutInflater inflater, @Nullable ViewGroup container,  
 @Nullable Bundle savedInstanceState) {  
 return inflater.inflate(R.layout.*fragment\_count*, container, false);  
 }  
  
 @Override  
 public void onViewCreated(@NonNull View view, @Nullable Bundle savedInstanceState) {  
 super.onViewCreated(view, savedInstanceState);  
  
 cancel = view.findViewById(R.id.*cancel*);  
 cancel.setOnClickListener(this::onCancel);  
  
 count = view.findViewById(R.id.*count*);  
 count.setOnClickListener(this::onCount);  
  
  
 type = view.findViewById(R.id.*type*);  
  
 setDataToTypes();  
 }  
  
 private void onCancel(View view) {  
 getParentFragmentManager().popBackStack();  
 }  
  
 private void onCount(View view) {  
 for (int i = 0; i < currentFigures.size(); i++) {  
 if (currentFigures.get(i).equals(type.getSelectedItem())) {  
 callback.handleEvent(new CountData(i, countType));  
 onCancel(view);  
 return;  
 }  
 }  
 }  
  
 private void setDataToTypes() {  
 ArrayAdapter<String> adapter = new ArrayAdapter<>(this.getContext(),  
 R.layout.*small\_spinner\_item*,  
 currentFigures);  
  
 adapter.setDropDownViewResource(android.R.layout.*simple\_spinner\_dropdown\_item*);  
 type.setAdapter(adapter);  
 }  
}

DragFigureFragment.java:

package com.example.geometry.views;  
  
import android.os.Bundle;  
import android.text.InputType;  
import android.view.LayoutInflater;  
import android.view.View;  
import android.view.ViewGroup;  
import android.widget.AdapterView;  
import android.widget.ArrayAdapter;  
import android.widget.Button;  
import android.widget.EditText;  
import android.widget.LinearLayout;  
import android.widget.Spinner;  
import android.widget.Toast;  
  
import androidx.annotation.NonNull;  
import androidx.annotation.Nullable;  
import androidx.fragment.app.Fragment;  
  
import com.example.geometry.R;  
import com.example.geometry.callbacks.DragFigureData;  
import com.example.geometry.utils.ICallbackHandler;  
  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Objects;  
  
public class DragFigureFragment extends Fragment {  
  
 private Button cancel = null;  
 private Button drag = null;  
 private Spinner type = null;  
 private Spinner dragType = null;  
 private LinearLayout container = null;  
 private final ICallbackHandler<DragFigureData> callback;  
  
 private final List<String> currentFigures;  
  
 //additional fields to add for specific move  
 private EditText xField = null;  
 private EditText yField = null;  
 private Spinner axis = null;  
  
 private DragFigureFragment(ICallbackHandler<DragFigureData> callback,  
 List<String> currentFigures) {  
 this.callback = callback;  
 this.currentFigures = currentFigures;  
 }  
  
 public static DragFigureFragment createInstance(ICallbackHandler<DragFigureData> callback,  
 List<String> currentFigures) {  
 return new DragFigureFragment(callback, currentFigures);  
 }  
  
 @Override  
 @Nullable  
 public View onCreateView(@NonNull LayoutInflater inflater, @Nullable ViewGroup container,  
 @Nullable Bundle savedInstanceState) {  
 return inflater.inflate(R.layout.*fragment\_drag\_figure*, container, false);  
 }  
  
 @Override  
 public void onViewCreated(@NonNull View view, @Nullable Bundle savedInstanceState) {  
 super.onViewCreated(view, savedInstanceState);  
  
 cancel = view.findViewById(R.id.*cancel*);  
 cancel.setOnClickListener(this::onCancel);  
  
 drag = view.findViewById(R.id.*drag*);  
 drag.setOnClickListener(this::onDragFigure);  
  
  
 type = view.findViewById(R.id.*type*);  
 dragType = view.findViewById(R.id.*dragType*);  
 container = view.findViewById(R.id.*container*);  
 axis = new Spinner(getContext());  
 xField = new EditText(getContext());  
 yField = new EditText(getContext());  
  
 setDataToTypes();  
 setDataToCombo();  
 setAxisSpinner();  
 setTextFields();  
 }  
  
 private void setTextFields() {  
 xField.setSingleLine();  
 xField.setInputType(InputType.*TYPE\_CLASS\_NUMBER*);  
 yField.setSingleLine();  
 yField.setInputType(InputType.*TYPE\_CLASS\_NUMBER*);  
 }  
  
 private void onCancel(View view) {  
 getParentFragmentManager().popBackStack();  
 }  
  
 private void onDragFigure(View view) {  
 String figure = (String) type.getSelectedItem();  
 String type = (String) dragType.getSelectedItem();  
  
 if (Objects.*equals*(figure, "") || Objects.*equals*(type, "")  
 || figure == null || type == null) {  
 showAlert("Incorrect data selected");  
 return;  
 }  
  
 for (int i = 0; i < currentFigures.size(); i++) {  
 if (currentFigures.get(i).equals(figure)) {  
 List<Double> args = new ArrayList<>();  
 if (type.equals("SymAxis")) {  
 args.add(Double.*parseDouble*((String) axis.getSelectedItem()));  
 } else if (type.equals("Rot")) {  
 args.add(Double.*parseDouble*(xField.getText().toString()));  
 } else {  
 args.add(Double.*parseDouble*(xField.getText().toString()));  
 args.add(Double.*parseDouble*(yField.getText().toString()));  
 }  
 callback.handleEvent(new DragFigureData(i, type, args));  
 onCancel(view);  
 return;  
 }  
 }  
  
 showAlert("Invalid data state");  
 }  
  
 private void changeLayoutOnSelectedDrag(String dragType) {  
 clearLayout();  
  
 if (Objects.*equals*(dragType, "Rot")) {  
 addFields(0);  
 } else if (Objects.*equals*(dragType, "SymAxis")) {  
 addAxisSpinner();  
 } else {  
 addFields(1);  
 }  
 container.requestLayout();  
 }  
  
 private void setDataToTypes() {  
 ArrayAdapter<String> adapter = new ArrayAdapter<>(this.getContext(),  
 R.layout.*small\_spinner\_item*,  
 currentFigures);  
  
 adapter.setDropDownViewResource(android.R.layout.*simple\_spinner\_dropdown\_item*);  
 type.setAdapter(adapter);  
 }  
  
 private void setDataToCombo() {  
 ArrayAdapter<CharSequence> adapter = ArrayAdapter.*createFromResource*(  
 this.getContext(),  
 R.array.*drag\_types*,  
 android.R.layout.*simple\_list\_item\_1* );  
 adapter.setDropDownViewResource(android.R.layout.*simple\_spinner\_dropdown\_item*);  
 dragType.setAdapter(adapter);  
  
 dragType.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {  
 @Override  
 public void onItemSelected(AdapterView<?> adapterView, View view, int i, long l) {  
 String[] choose = getResources().getStringArray(R.array.*drag\_types*);  
 DragFigureFragment.this.changeLayoutOnSelectedDrag(choose[i]);  
 }  
  
 @Override  
 public void onNothingSelected(AdapterView<?> adapterView) {  
 // pass  
 }  
 });  
 }  
  
 private void setAxisSpinner() {  
 ArrayAdapter<CharSequence> adapter = ArrayAdapter.*createFromResource*(  
 this.getContext(),  
 R.array.*axis\_numbers*,  
 android.R.layout.*simple\_list\_item\_1* );  
 adapter.setDropDownViewResource(android.R.layout.*simple\_spinner\_dropdown\_item*);  
 axis.setAdapter(adapter);  
 }  
  
 private void clearLayout() {  
 container.removeAllViews();  
 }  
  
 private void addFields(int combination) {  
 container.addView(xField);  
 if (combination == 1) {  
 container.addView(yField);  
 }  
 }  
  
 private void addAxisSpinner() {  
 if (axis != null) {  
 container.addView(axis);  
 }  
 }  
  
 private void showAlert(CharSequence text) {  
 Toast.*makeText*(getContext(),  
 text,  
 Toast.*LENGTH\_SHORT*).show();  
 }  
}

MainActivity.java:

package com.example.geometry.views;  
  
import androidx.appcompat.app.AppCompatActivity;  
import androidx.fragment.app.Fragment;  
import androidx.fragment.app.FragmentContainerView;  
  
import android.os.Bundle;  
  
import com.example.geometry.R;  
  
public class MainActivity extends AppCompatActivity {  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*activity\_main*);  
  
 setMainFragment();  
 }  
  
 public void navigate(Class<? extends Fragment> to) {  
 getSupportFragmentManager().beginTransaction()  
 .replace(R.id.*container*, to, null)  
 .setReorderingAllowed(true)  
 .addToBackStack(to.getName())  
 .commit();  
 }  
  
 public void navigate(Fragment to) {  
 getSupportFragmentManager().beginTransaction()  
 .replace(R.id.*container*, to, to.getClass().toString())  
 .setReorderingAllowed(true)  
 .addToBackStack(to.getClass().toString())  
 .commit();  
 }  
  
 private void setMainFragment() {  
 getSupportFragmentManager().beginTransaction()  
 .add(R.id.*container*, MainFragment.class, null)  
 .setReorderingAllowed(true)  
 .commit();  
 }  
}

MainFragment.java:

package com.example.geometry.views;  
  
import android.content.ContentValues;  
import android.content.Context;  
import android.graphics.Bitmap;  
import android.graphics.Canvas;  
import android.net.Uri;  
import android.os.Bundle;  
import android.provider.MediaStore;  
import android.util.Log;  
import android.view.LayoutInflater;  
import android.view.View;  
import android.view.ViewGroup;  
import android.widget.Button;  
import android.widget.TextView;  
import android.widget.Toast;  
  
import androidx.activity.result.ActivityResultLauncher;  
import androidx.activity.result.contract.ActivityResultContracts;  
import androidx.annotation.NonNull;  
import androidx.annotation.Nullable;  
import androidx.fragment.app.Fragment;  
  
import com.example.common\_utils.deserializers.IFigureFactory;  
import com.example.common\_utils.repository.MongoDb;  
import com.example.geometry.R;  
import com.example.geometry.callbacks.AddFigureData;  
import com.example.geometry.callbacks.CheckIfCrossData;  
import com.example.geometry.callbacks.CountData;  
import com.example.geometry.callbacks.DragFigureData;  
import com.example.geometry.callbacks.RemoveData;  
import com.example.geometry.utils.GeometryCanvas;  
  
import java.io.BufferedReader;  
import java.io.FileNotFoundException;  
import java.io.IOException;  
import java.io.InputStream;  
import java.io.InputStreamReader;  
import java.io.OutputStream;  
import java.nio.charset.StandardCharsets;  
import java.util.ArrayList;  
import java.util.List;  
  
import app.Circle;  
import app.IShape;  
import app.NGon;  
import app.Point2D;  
import app.Polyline;  
import app.QGon;  
import app.Rectangle;  
import app.Segment;  
import app.TGon;  
import app.Trapeze;  
  
public class MainFragment extends Fragment {  
  
 private static final String *fileName* = "./src/main/java/ui/figures.txt";  
 private static final String *pngFilePath* = "./src/main/java/ui/snapshot.png";  
  
 public List<IShape> list = new ArrayList<>();  
 public List<String> types = new ArrayList<>();  
  
 private Button addFigure = null;  
 private Button removeFigure = null;  
 private Button saveAsImg = null;  
 private Button saveToFile = null;  
 private Button computeS = null;  
 private Button computeP = null;  
 private Button dragFigure = null;  
 private Button addToDB = null;  
 private Button retrieveFromDB = null;  
 private Button clear = null;  
 private Button uploadFromFile = null;  
 private Button checkIfCross = null;  
  
 private GeometryCanvas canvas = null;  
 private TextView answer = null;  
  
 private MongoDb database = null;  
  
 ActivityResultLauncher<String[]> readFileLauncher = registerForActivityResult(  
 new ActivityResultContracts.OpenDocument(),  
 this::readFromDocument  
 );  
  
 ActivityResultLauncher<String[]> writeToFileLauncher = registerForActivityResult(  
 new ActivityResultContracts.OpenDocument(),  
 this::writeToDocument  
 );  
  
 @Override  
 @Nullable  
 public View onCreateView(@NonNull LayoutInflater inflater, @Nullable ViewGroup container,  
 @Nullable Bundle savedInstanceState) {  
 return inflater.inflate(R.layout.*fragment\_main*, container, false);  
 }  
  
 @Override  
 public void onViewCreated(@NonNull View view, @Nullable Bundle savedInstanceState) {  
 super.onViewCreated(view, savedInstanceState);  
 database = MongoDb.*create*();  
  
 canvas = view.findViewById(R.id.*geometry*);  
 answer = view.findViewById(R.id.*resultField*);  
  
 addFigure = view.findViewById(R.id.*addFigure*);  
 addFigure.setOnClickListener(this::onAddFigure);  
  
 removeFigure = view.findViewById(R.id.*removeFigure*);  
 removeFigure.setOnClickListener(this::onRemoveFigure);  
  
 saveAsImg = view.findViewById(R.id.*saveAsImg*);  
 saveAsImg.setOnClickListener(this::onSaveAsImg);  
  
 saveToFile = view.findViewById(R.id.*saveToFile*);  
 saveToFile.setOnClickListener(this::writeToFile);  
  
 computeS = view.findViewById(R.id.*computeS*);  
 computeS.setOnClickListener(this::onComputeS);  
  
 computeP = view.findViewById(R.id.*computeP*);  
 computeP.setOnClickListener(this::onComputeP);  
  
 dragFigure = view.findViewById(R.id.*dragFigure*);  
 dragFigure.setOnClickListener(this::onDragFigure);  
  
 addToDB = view.findViewById(R.id.*addToDB*);  
 addToDB.setOnClickListener(this::onAddToDatabase);  
  
 retrieveFromDB = view.findViewById(R.id.*retrieveFromDB*);  
 retrieveFromDB.setOnClickListener(this::onRetrieveFromDatabase);  
  
 clear = view.findViewById(R.id.*clear*);  
 clear.setOnClickListener(this::onInitialize);  
  
 uploadFromFile = view.findViewById(R.id.*uploadFromFile*);  
 uploadFromFile.setOnClickListener(this::onUploadFromFile);  
  
 checkIfCross = view.findViewById(R.id.*checkIfCross*);  
 checkIfCross.setOnClickListener(this::onCheckIfCross);  
 }  
  
 public static void showAlert(Context context, String alert) {  
 Toast.*makeText*(context, alert, Toast.*LENGTH\_SHORT*).show();  
 }  
  
 // callback func  
 public void moveFigure(DragFigureData data) {  
 int pos = data.getIndex();  
 String type = data.getType().trim();  
 Double[] args = data.getCoords().toArray(new Double[0]);  
  
 IShape shape = canvas.getList().get(pos);  
 canvas.remove(pos);  
  
 try {  
 switch (type) {  
 case "Rot":  
 shape.rot(args[0]);  
 break;  
 case "SymAxis":  
 shape.symAxis(args[0].intValue());  
 break;  
 case "Shift":  
 shape.shift(new Point2D(args[0], args[1]));  
 break;  
 }  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
  
 canvas.addToListAndDraw(shape, false);  
 }  
  
 // callback func  
 public void countByPos(CountData data) {  
 int pos = data.getFirst();  
 boolean type = data.getType();  
 canvas.addToListAndDraw(list.get(pos), false);  
  
 answer.setVisibility(View.*VISIBLE*);  
 double res = 0.0;  
 try {  
 if (type)  
 res = list.get(pos).length();  
 else  
 res = list.get(pos).square();  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
  
 answer.setText(String.*valueOf*(res));  
 }  
  
 // callback func  
 public void setIfCross(CheckIfCrossData data) {  
 int firstPosition = data.getFirst();  
 int secondPosition = data.getSecond();  
  
 try {  
 answer.setVisibility(View.*VISIBLE*);  
 if (canvas.getList().get(firstPosition).cross(canvas.getList().get(secondPosition)))  
 answer.setText("Cross");  
 else  
 answer.setText("Not Cross");  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
  
 // callback func  
 public void removeFigureByPosition(RemoveData data) {  
 int pos = data.getPosition();  
 canvas.remove(pos);  
 canvas.clearAndDrawAll(null, false);  
 }  
  
 public void readFromDocument(Uri uri) {  
 InputStream fstream;  
 try {  
 fstream = requireContext().getContentResolver().openInputStream(uri);  
 } catch (FileNotFoundException e) {  
 e.printStackTrace();  
 return;  
 }  
  
 BufferedReader br = new BufferedReader(new InputStreamReader(fstream));  
 try {  
 String line;  
 int lineInd = 0;  
 while ((line = br.readLine()) != null) {  
 Log.*e*("data", line);  
 IShape restoredShape = IFigureFactory.*create*(line);  
 if (restoredShape != null) {  
 canvas.addToListAndDraw(restoredShape, true);  
 } else {  
 System.*out*.println(String.*format*("Shape on line %d is invalid, ignoring it", lineInd));  
 }  
 lineInd++;  
 }  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 try {  
 br.close();  
 fstream.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 public void writeToDocument(Uri uri) {  
 try {  
 OutputStream stream = getContext().getContentResolver().openOutputStream(uri, "w");  
 String endOf = "\n";  
 for (int i = 0; i < canvas.getList().size(); i++) {  
 if (i == canvas.getList().size() - 1)  
 endOf = "";  
 stream.write(  
 String.*format*("%s%s", canvas.getList().get(i).toString(), endOf)  
 .getBytes(StandardCharsets.*UTF\_8*));  
 }  
 stream.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 public void addCircle(double[] coords) throws Exception {  
 double r = coords[2];  
 Point2D point = new Point2D(coords[0], coords[1]);  
 canvas.addToListAndDraw(new Circle(point, r), false);  
 }  
  
 public void addNGon(double[] coords) throws Exception {  
 int n = coords.length / 2;  
 Point2D[] points = new Point2D[n];  
 for (int i = 0; i < coords.length; i += 2) {  
 points[i / 2] = new Point2D(coords[i], coords[i + 1]);  
 }  
  
 canvas.addToListAndDraw(new NGon(points), false);  
 }  
  
 public void addQGon(double[] coords) throws Exception {  
 Point2D[] points = new Point2D[4];  
 for (int i = 0; i < coords.length; i += 2) {  
 points[i / 2] = new Point2D(coords[i], coords[i + 1]);  
 }  
 canvas.addToListAndDraw(new QGon(points), false);  
 }  
  
 public void addRectangle(double[] coords) throws Exception {  
 Point2D[] points = new Point2D[4];  
 for (int i = 0; i < coords.length; i += 2) {  
 points[i / 2] = new Point2D(coords[i], coords[i + 1]);  
 }  
 canvas.addToListAndDraw(new Rectangle(points), false);  
 }  
  
 public void addTrapeze(double[] coords) throws Exception {  
 Point2D[] points = new Point2D[4];  
 for (int i = 0; i < coords.length; i += 2) {  
 points[i / 2] = new Point2D(coords[i], coords[i + 1]);  
 }  
 canvas.addToListAndDraw(new Trapeze(points), false);  
 }  
  
 public void addPolyline(double[] coords) throws Exception {  
 int n = coords.length / 2;  
 Point2D[] points = new Point2D[n];  
 for (int i = 0; i < coords.length; i += 2) {  
 points[i / 2] = new Point2D(coords[i], coords[i + 1]);  
 }  
 canvas.addToListAndDraw(new Polyline(points), false);  
 }  
  
 public void addTGon(double[] coords) throws Exception {  
 int n = 3;  
 Point2D[] points = new Point2D[n];  
 for (int i = 0; i < coords.length; i += 2) {  
 points[i / 2] = new Point2D(coords[i], coords[i + 1]);  
 }  
  
 canvas.addToListAndDraw(new TGon(points), false);  
 }  
  
 public void addSegment(double[] coords) throws Exception {  
 canvas.addToListAndDraw(new Segment(new Point2D(coords[0], coords[1]),  
 new Point2D(coords[2], coords[3])), false);  
 }  
  
 public void addShape(AddFigureData data) throws Exception {  
 String type = data.getType();  
 double[] coords = new double[data.getCoords().size()];  
 for (int i = 0; i < coords.length; i++) {  
 coords[i] = data.getCoords().get(i);  
 }  
 switch (type) {  
 case "Circle":  
 addCircle(coords);  
 break;  
 case "Segment":  
 addSegment(coords);  
 break;  
 case "TGon":  
 addTGon(coords);  
 break;  
 case "NGon":  
 addNGon(coords);  
 break;  
 case "QGon":  
 addQGon(coords);  
 break;  
 case "Trapeze":  
 addTrapeze(coords);  
 break;  
 case "Rectangle":  
 addRectangle(coords);  
 break;  
 case "Polyline":  
 addPolyline(coords);  
 break;  
 }  
 }  
  
 private void onAddFigure(View view) {  
 ((MainActivity) requireActivity()).navigate(AddFigureFragment.*createInstance*((data) -> {  
 try {  
 MainFragment.this.addShape(data);  
 } catch (Exception e) {  
 Log.*e*(MainFragment.class.toString(), e.getMessage());  
 }  
 }));  
 }  
  
 private void onInitialize(View view) {  
 canvas.removeAll(true);  
 }  
  
 private void onUploadFromFile(View view) {  
 readFileLauncher.launch(new String[]{"text/plain"});  
 }  
  
 public Bitmap viewToBitmap(View view) {  
 Bitmap bitmap = Bitmap.*createBitmap*(view.getWidth(), view.getHeight(), Bitmap.Config.*ARGB\_8888*);  
 Canvas canvas = new Canvas(bitmap);  
 view.draw(canvas);  
 return bitmap;  
 }  
  
 private void onSaveAsImg(View view) {  
 Bitmap bitmap = viewToBitmap(canvas);  
 ContentValues values = contentValues();  
 values.put(MediaStore.Images.Media.*RELATIVE\_PATH*, "Pictures/Geometry");  
 values.put(MediaStore.Images.Media.*IS\_PENDING*, true);  
  
 Uri uri = getContext().getContentResolver().insert(MediaStore.Images.Media.*EXTERNAL\_CONTENT\_URI*, values);  
 if (uri != null) {  
 try {  
 saveImageToStream(bitmap, getContext().getContentResolver().openOutputStream(uri));  
 } catch (FileNotFoundException e) {  
 e.printStackTrace();  
 }  
 values.put(MediaStore.Images.Media.*IS\_PENDING*, false);  
 getContext().getContentResolver().update(uri, values, null, null);  
 }  
 }  
  
 private ContentValues contentValues() {  
 ContentValues values = new ContentValues();  
 values.put(MediaStore.Images.Media.*MIME\_TYPE*, "image/png");  
 values.put(MediaStore.Images.Media.*DATE\_ADDED*, System.*currentTimeMillis*() / 1000);  
 values.put(MediaStore.Images.Media.*DATE\_TAKEN*, System.*currentTimeMillis*());  
 return values;  
 }  
  
 private void saveImageToStream(Bitmap bitmap, OutputStream outputStream) {  
 if (outputStream != null) {  
 try {  
 bitmap.compress(Bitmap.CompressFormat.*PNG*, 100, outputStream);  
 outputStream.close();  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
 }  
  
 private void onDragFigure(View view) {  
 List<String> stringList = new ArrayList<>();  
 canvas.getList().forEach((shape) -> {  
 stringList.add(shape.toString());  
 });  
 ((MainActivity) requireActivity()).navigate(DragFigureFragment.*createInstance*((data) -> {  
 try {  
 MainFragment.this.moveFigure(data);  
 } catch (Exception e) {  
 Log.*e*(MainFragment.class.toString(), e.getMessage());  
 }  
 }, stringList));  
 }  
  
 private void onRemoveFigure(View view) {  
 List<String> stringList = new ArrayList<>();  
 canvas.getList().forEach((shape) -> {  
 stringList.add(shape.toString());  
 });  
 ((MainActivity) requireActivity()).navigate(RemoveFragment.*createInstance*((data) -> {  
 try {  
 MainFragment.this.removeFigureByPosition(data);  
 } catch (Exception e) {  
 Log.*e*(MainFragment.class.toString(), e.getMessage());  
 }  
 }, stringList));  
 }  
  
 private void onComputeS(View view) {  
 List<String> stringList = new ArrayList<>();  
 canvas.getList().forEach((shape) -> {  
 stringList.add(shape.toString());  
 });  
 ((MainActivity) requireActivity()).navigate(CountFragment.*createInstance*((data) -> {  
 try {  
 MainFragment.this.countByPos(data);  
 } catch (Exception e) {  
 Log.*e*(MainFragment.class.toString(), e.getMessage());  
 }  
 }, stringList, false));  
 }  
  
 private void onComputeP(View view) {  
 List<String> stringList = new ArrayList<>();  
 canvas.getList().forEach((shape) -> {  
 stringList.add(shape.toString());  
 });  
 ((MainActivity) requireActivity()).navigate(CountFragment.*createInstance*((data) -> {  
 try {  
 MainFragment.this.countByPos(data);  
 } catch (Exception e) {  
 Log.*e*(MainFragment.class.toString(), e.getMessage());  
 }  
 }, stringList, true));  
 }  
  
 private void onCheckIfCross(View view) {  
 List<String> stringList = new ArrayList<>();  
 canvas.getList().forEach((shape) -> {  
 stringList.add(shape.toString());  
 });  
 ((MainActivity) requireActivity()).navigate(CheckIfCrossFragment.*createInstance*((data) -> {  
 try {  
 MainFragment.this.setIfCross(data);  
 } catch (Exception e) {  
 Log.*e*(MainFragment.class.toString(), e.getMessage());  
 }  
 }, stringList));  
 }  
  
 private void onAddToDatabase(View view) {  
 database.addToDatabase(canvas.getList(), null);  
 }  
  
 private void onRetrieveFromDatabase(View view) {  
 database.retrieveFromDatabase((restoredList) -> {  
 MainFragment.this.requireActivity().runOnUiThread(() -> {  
 restoredList.forEach((shape) -> {  
 canvas.addToListAndDraw(shape, true);  
 });  
 });  
 });  
 }  
  
 private void writeToFile(View view) {  
 writeToFileLauncher.launch(new String[]{"text/plain"});  
 }  
  
 @Override  
 public void onDestroyView() {  
 addFigure = null;  
 removeFigure = null;  
 saveAsImg = null;  
 saveToFile = null;  
 computeS = null;  
 computeP = null;  
 dragFigure = null;  
 addToDB = null;  
 retrieveFromDB = null;  
 clear = null;  
 uploadFromFile = null;  
 checkIfCross = null;  
 super.onDestroyView();  
 }  
}

RemoveFragment.java:

package com.example.geometry.views;  
  
import android.os.Bundle;  
import android.view.LayoutInflater;  
import android.view.View;  
import android.view.ViewGroup;  
import android.widget.ArrayAdapter;  
import android.widget.Button;  
import android.widget.Spinner;  
  
import androidx.annotation.NonNull;  
import androidx.annotation.Nullable;  
import androidx.fragment.app.Fragment;  
  
import com.example.geometry.R;  
import com.example.geometry.callbacks.RemoveData;  
import com.example.geometry.utils.ICallbackHandler;  
  
import java.util.List;  
  
import app.IShape;  
  
public class RemoveFragment extends Fragment {  
  
 private final ICallbackHandler<RemoveData> callback;  
 private final List<String> currentFigures;  
  
 private Button cancel = null;  
 private Button remove = null;  
 private Spinner type = null;  
  
 private RemoveFragment(ICallbackHandler<RemoveData> callback,  
 List<String> currentFigures) {  
 this.callback = callback;  
 this.currentFigures = currentFigures;  
 }  
  
 public static RemoveFragment createInstance(ICallbackHandler<RemoveData> callback,  
 List<String> currentFigures) {  
 return new RemoveFragment(callback, currentFigures);  
 }  
  
 @Override  
 @Nullable  
 public View onCreateView(@NonNull LayoutInflater inflater, @Nullable ViewGroup container,  
 @Nullable Bundle savedInstanceState) {  
 return inflater.inflate(R.layout.*fragment\_remove*, container, false);  
 }  
  
 @Override  
 public void onViewCreated(@NonNull View view, @Nullable Bundle savedInstanceState) {  
 super.onViewCreated(view, savedInstanceState);  
  
 cancel = view.findViewById(R.id.*cancel*);  
 cancel.setOnClickListener(this::onCancel);  
  
 remove = view.findViewById(R.id.*remove*);  
 remove.setOnClickListener(this::onRemove);  
  
  
 type = view.findViewById(R.id.*type*);  
  
 setDataToTypes();  
 }  
  
 private void onCancel(View view) {  
 getParentFragmentManager().popBackStack();  
 }  
  
 private void onRemove(View view) {  
 String obj = (String) type.getSelectedItem();  
 for (int i = 0; i < currentFigures.size(); i++) {  
 if (currentFigures.get(i).equals(obj)) {  
 callback.handleEvent(new RemoveData(i));  
 onCancel(view);  
 return;  
 }  
 }  
 }  
  
 private void setDataToTypes() {  
 ArrayAdapter<String> adapter = new ArrayAdapter<>(this.getContext(),  
 R.layout.*small\_spinner\_item*,  
 currentFigures);  
  
 adapter.setDropDownViewResource(android.R.layout.*simple\_spinner\_dropdown\_item*);  
 type.setAdapter(adapter);  
 }  
}

build.gradle:

// Top-level build file where you can add configuration options common to all sub-projects/modules.  
  
buildscript **{** repositories **{** mavenCentral()  
 **}** dependencies **{** classpath "io.realm:realm-gradle-plugin:10.13.0"  
 **}  
}**plugins **{** id 'com.android.application' version '7.2.2' apply false  
 id 'com.android.library' version '7.2.2' apply false  
**}**ext **{** versions = [  
 material\_version : "1.6.1",  
 dataStore : "1.0.0-rc01",  
 lifecycle\_ver : "2.4.0-rc01",  
 fragment : "1.4.0-alpha04",  
 gson\_version : "2.8.8",  
 retrofit2\_version : "2.9.0",  
 room\_version : "2.3.0",  
 hilt\_version : "2.38.1",  
 okhttp\_version : "3.14.6",  
 adapter\_delegate : "4.3.2",  
 core\_ktx\_version : "1.7.0",  
 appcompat\_version : "1.3.1",  
 constraintlayout\_version : "2.1.4",  
 junit\_version : "4.13.2",  
 junit\_ext\_version : "1.1.3",  
 espresso\_core\_version : "3.4.0",  
 swiperefreshlayout\_version: "1.1.0",  
 mongo\_driver\_version : "4.1.2",  
 awt\_version : "1.0.0",  
 lombok\_version : "1.18.4",  
 smack\_version : "4.4.6",  
 realm\_version : "10.13.0",  
 ]  
**}**def libs(libname) {  
 switch (libname) {  
 case "material":  
 return "com.google.android.material:material:$**{**versions.material\_version**}**"  
 case "core-ktx":  
 return "androidx.core:core-ktx:$**{**versions.core\_ctx\_version**}**"  
 case "appcompat":  
 return "androidx.appcompat:appcompat:$**{**versions.appcompat\_version**}**"  
 case "constraintlayout":  
 return "androidx.constraintlayout:constraintlayout:$**{**versions.constraintlayout\_version**}**"  
 case "junit":  
 return "junit:junit:$**{**versions.junit\_version**}**"  
 case "junit-ext":  
 return "androidx.test.ext:junit:$**{**versions.junit\_ext\_version**}**"  
 case "espresso":  
 return "androidx.test.espresso:espresso-core:$**{**versions.espresso\_version**}**"  
 case "swiperefreshlayout":  
 return "androidx.swiperefreshlayout:swiperefreshlayout:$**{**versions.swiperefreshlayout\_version**}**"  
 case "datastore":  
 return "androidx.datastore:datastore-preferences:$**{**versions.dataStore**}**"  
 case "lifecycle-runtime":  
 return "androidx.lifecycle:lifecycle-runtime:$**{**versions.lifecycle\_ver**}**"  
 case "lifecycle-compiler":  
 return "androidx.lifecycle:lifecycle-compiler:$**{**versions.lifecycle\_ver**}**"  
 case "fragment-ktx":  
 return "androidx.fragment:fragment-ktx:$**{**versions.fragment**}**"  
 case "viewmodel":  
 return "androidx.lifecycle:lifecycle-viewmodel-ktx:$**{**versions.lifecycle\_ver**}**"  
 case "gson":  
 return "com.google.code.gson:gson:$**{**versions.gson\_version**}**"  
 case "retrofit":  
 return "com.squareup.retrofit2:retrofit:$**{**versions.retrofit2\_version**}**"  
 case "retrofit-converter":  
 return "com.squareup.retrofit2:converter-gson:$**{**versions.retrofit2\_version**}**"  
 case "room":  
 return "androidx.room:room-runtime:$**{**versions.room\_version**}**"  
 case "room-compiler":  
 return "androidx.room:room-compiler:$**{**versions.room\_version**}**"  
 case "room-ktx":  
 return "androidx.room:room-ktx:$**{**versions.room\_version**}**"  
 case "dagger":  
 return "com.google.dagger:hilt-android:$**{**versions.hilt\_version**}**"  
 case "hilt":  
 return "com.google.dagger:hilt-compiler:$**{**versions.hilt\_version**}**"  
 case "okhttp3":  
 return "com.squareup.okhttp3:okhttp:$**{**versions.okhttp\_version**}**"  
 case "adapterdelegate":  
 return "com.hannesdorfmann:adapterdelegates4-kotlin-dsl-viewbinding:$**{**versions.adapter\_delegate**}**"  
 case "mongodb":  
 return "org.mongodb:mongodb-driver-sync:$**{**versions.mongo\_driver\_version**}**"  
 case "awt":  
 return "clojure-interop:java.awt:$**{**versions.awt\_version**}**"  
 case "lombok":  
 return "org.projectlombok:lombok:$**{**versions.lombok\_version**}**"  
 case "smack-android":  
 return "org.igniterealtime.smack:smack-android-extensions:$**{**versions.smack\_version**}**"  
 case "smack-experimental":  
 return "org.igniterealtime.smack:smack-experimental:$**{**versions.smack\_version**}**"  
 case "smack-tcp":  
 return "org.igniterealtime.smack:smack-tcp:$**{**versions.smack\_version**}**"  
 case "realm":  
 return "io.realm:realm-gradle-plugin:$**{**versions.realm\_version**}**"  
 default:  
 throw new IllegalArgumentException("No mapping exists for name: $name.")  
 }  
}  
  
task clean(type: Delete) **{** delete rootProject.buildDir  
**}**

plugins **{** id 'com.android.application'  
**}**android **{** compileSdk 32  
  
 defaultConfig **{** applicationId "com.example.geometry"  
 minSdk 29  
 targetSdk 32  
 versionCode 1  
 versionName "1.0"  
  
 testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"  
 **}** buildTypes **{** release **{** minifyEnabled false  
 proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'  
 **}  
 }** compileOptions **{** sourceCompatibility JavaVersion.*VERSION\_11* targetCompatibility JavaVersion.*VERSION\_11* **}  
}**dependencies **{** implementation project(":common-utils")  
  
 implementation libs("lifecycle-compiler")  
 implementation libs("viewmodel")  
 implementation libs("core-ktx")  
 implementation libs("appcompat")  
 implementation libs("material")  
 implementation libs("constraintlayout")  
 implementation libs("lifecycle-runtime")  
 implementation libs("fragment-ktx")  
  
 testImplementation libs("junit")  
 androidTestImplementation libs("junit-ext")  
 androidTestImplementation libs("espresso")  
**}**

plugins **{** id 'com.android.library'  
**}**apply plugin: "realm-android"  
  
android {  
 compileSdk 32  
  
 defaultConfig {  
 minSdk 29  
 targetSdk 32  
 versionCode 1  
 versionName "1.0"  
  
 testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"  
 }  
  
 buildTypes {  
 release {  
 minifyEnabled false  
 proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'  
 }  
 }  
 compileOptions {  
 sourceCompatibility JavaVersion.VERSION\_11  
 targetCompatibility JavaVersion.VERSION\_11  
 }  
}  
  
realm {  
 syncEnabled = true  
}  
  
dependencies {  
 api project(":data-module")  
 implementation libs("appcompat")  
 implementation libs("material")  
 implementation libs("mongodb")  
  
 implementation libs("realm")  
}

plugins {  
 id 'java-library'  
}  
  
java {  
 sourceCompatibility = JavaVersion.VERSION\_11  
 targetCompatibility = JavaVersion.VERSION\_11  
}  
  
dependencies {  
 implementation libs("mongodb")  
}

pluginManagement **{** repositories **{** gradlePluginPortal()  
 google()  
 mavenCentral()  
 **}  
}**dependencyResolutionManagement {  
 repositoriesMode.set(RepositoriesMode.FAIL\_ON\_PROJECT\_REPOS)  
 repositories {  
 google()  
 mavenCentral()  
 }  
}  
rootProject.name = "JavaGeometryExample"  
include ':app'  
include ':data-module'  
include ':common-utils'