**Комитет по образованию г. Санкт-Петербург**

**ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБЩЕОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ**

**ПРЕЗИДЕНТСКИЙ ФИЗИКО-МАТЕМАТИЧЕСКИЙ**

**ЛИЦЕЙ №239**

**Отчет о практике**

**«Создание графических приложений на языке Java»**

Учащаяся 10-1 класса

Григина У.А.

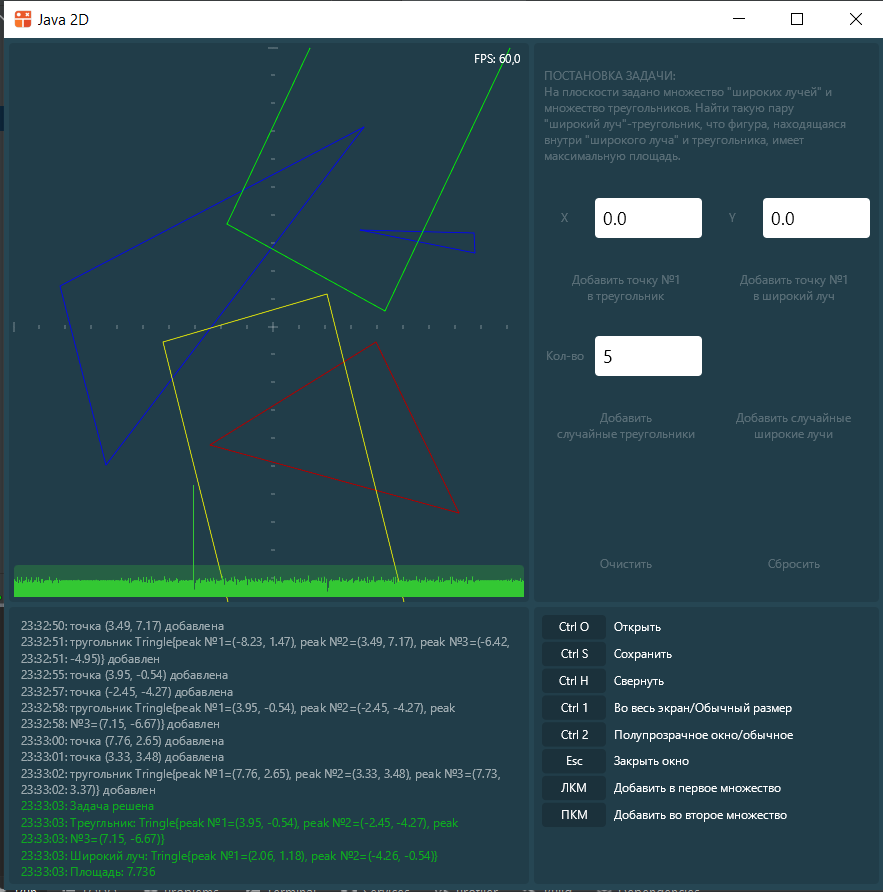
Преподаватель:

Клюнин А.О.

Санкт-Петербург – 2023 год

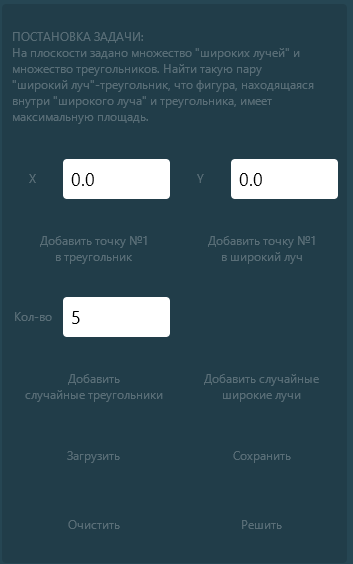
# 1. Постановка задачи

На плоскости задано множество "широких лучей" и множество треугольников. Найти такую пару "широкий луч"-треугольник, что фигура, находящаяся внутри "широкого луча" и треугольника, имеет максимальную площадь. В качестве ответа: выделить найденные "широкий луч" и треугольник.



# 2. Элементы управления

В рамках данной задачи необходимо было реализовать следующие элементы управления:



Для добавления треугольников и широких лучей были созданы 2 поля ввода (для **X** и **Y** координат) и 2 кнопки. Фигуры задаются по введённым точкам. При нажатии на первую кнопку точка добавляется в треугольник, при нажатии на вторую – в широкий луч.

Так же было создано 1 поле для ввода количества случайных фигур и 2 кнопки. При нажатии на первую кнопку добавляются случайные треугольники, при нажатии на вторую – в широкие лучи.

Созданы кнопки для загрузки и сохранении состояния программы.

Так же есть кнопки для решения/сброса решения задачи и очистки данных.

# 3. Структуры данных

Для хранения треугольников разработан класс **Triangle.java**, для широких лучей – **Beam.java**

Треугольник содержит поля **peaks**, хранящий список вершин треугольника, и **S**, хранящий площадь треугольника.

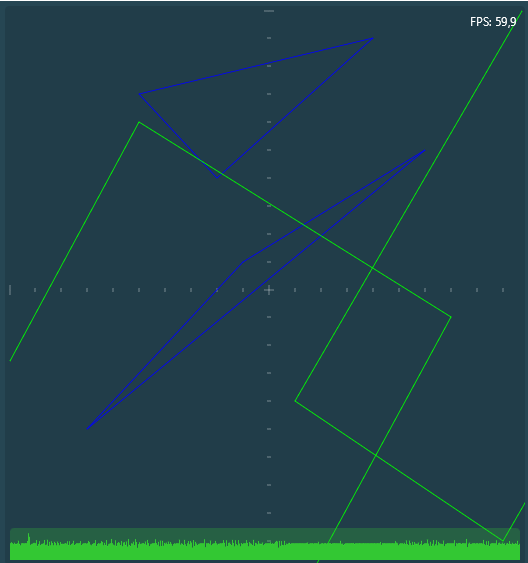
Широкий луч содержит поле **peaks**, хранящее список вершин широкого луча в порядке их введения.

В них разработаны функции **isInside()** проверяющие находится ли точка внутри фигуры

Их листинги приведены в приложении А.

# 4. Рисование

Для рисования фигур использовалась функция **canvas.drawLine**(), а для точек – **canvas.drawPoint()**



# 5. Решение задачи

Для решения поставленной задачи в классе **Task** был разработан метод **solve().**

*/\*\*  
 \* Решить задачу  
 \*/*public void solve() {  
 // очищаем предыдущий ответ  
 cancel();  
  
 for (int i = 0; i < triangles.size(); ++i) {  
 for (int j = 0; j < beams.size(); ++j) {  
 double S = get\_S\_OfCross(triangles.get(i), beams.get(j));  
 if (S > maxS) {  
 maxS = S;  
 indexTriangle = i;  
 indexBeam = j;  
 }  
 }  
 }  
  
 // задача решена  
 solved = true;  
}

В нём перебираются все возможные пары треугольников и широких лучей и сравнивается площадь пересечений этих пар при помощи метода Монте-Карло, реализованного в функции **get\_S\_OfCross()**

*/\*\*  
 \* Получить примерную площадь пересечения треугольника и широкого луча  
 \** ***@param*** *t треугольник  
 \** ***@param*** *b широкий луч  
 \** ***@return*** *примерная площадь  
 \*/*private double get\_S\_OfCross(Triangle t, Beam b) {  
 int n = 100000;  
 int cnt = 0;  
 for (int i = 0; i < n; ++i) {  
 Vector2d pos = ownCS.getRandomCoords();  
 if (t.isInside(pos) && b.isInside(pos)) ++cnt;  
 }  
 return ownCS.getSize().x \* ownCS.getSize().y / n \* cnt;  
}

Ответ храниться в **indexTriangle** и **indexBeam**, указывающие на искомые треугольник и широкий луч в их соответствующих списках

# 6. Проверка

Для проверки правильности решённой задачи были разработаны unit-тесты. Их листинг приведён в приложении Б.

Тест «треугольника» (проверяет функцию **isInside()**)

Теугольник: ), (0, 4), (2, -2)}

Точки: }

Тест «широкого луча» (проверяет функцию **isInside()**)

Широкий луч: ), (2, 0)}

Точки: {(1, 0), (0, 0), (-3, -3), (5, 0), (-5, 0),(6, 6,) (6, 5)}

Тест решения (проверяет решение)

Треугольники: {(-7, -5),(-1, 1),(6, 5)}, {(-5, 7),(-2, 4),(4, 9)}

Широкие лучи: {(7, -1),(-5, 6)}, {(1, -4),(9, -9)}

Ответ: **indexBeam = 0; indexTriangle = 0**

# 7. Заключение

В рамках выполнения поставленной задачи было создано графическое приложение с требуемым функционалом. Правильность решения задачи проверена с помощью юнит-тестов.

# Приложение А. Структуры данных

package app;  
  
import com.fasterxml.jackson.annotation.JsonCreator;  
import com.fasterxml.jackson.annotation.JsonIgnore;  
import com.fasterxml.jackson.annotation.JsonProperty;  
import misc.Misc;  
import misc.Vector2d;  
  
import java.util.ArrayList;  
import java.util.List;  
  
*/\*\*  
 \* Класс треугольника  
 \*/*public class Triangle {  
 */\*\*  
 \* Координаты вершин треугольника  
 \*/* public final ArrayList<Vector2d> peaks;  
  
 */\*\*  
 \* Площать треугольника  
 \*/* public final double S;  
  
 */\*\*  
 \* Конструктор треугольника  
 \** ***@param*** *peaks вершины  
 \*/* @JsonCreator  
 public Triangle(@JsonProperty("peaks") ArrayList<Vector2d> peaks) {  
 this.peaks = peaks;  
 this.S = getSquareBy3Points(peaks.get(0), peaks.get(1), peaks.get(2));  
 }  
  
 */\*\*  
 \* Получить площадь триугольника по вершинам  
 \** ***@param*** *p1 вершина №1  
 \** ***@param*** *p2 вершина №2  
 \** ***@param*** *p3 вершина №3  
 \** ***@return*** *Площадь  
 \*/* double getSquareBy3Points(Vector2d p1, Vector2d p2, Vector2d p3) {  
 Vector2d v1 = Vector2d.*subtract*(p2, p1);  
 Vector2d v2 = Vector2d.*subtract*(p3, p1);  
 return (Math.*abs*(v1.x \* v2.y - v1.y \* v2.x)) / 2;  
 }  
  
 */\*\*  
 \* Получить цвет треугольника  
 \*  
 \** ***@return*** *цвет точки  
 \*/* public static int getColor() {  
 return Misc.*getColor*(0xCC, 0x00, 0x00, 0xFF);  
 }  
  
 */\*\*  
 \* Строковое представление объекта  
 \*  
 \** ***@return*** *строковое представление объекта  
 \*/* @Override  
 public String toString() {  
 return "Tringle{" +  
 "peak №1=" + peaks.get(0) +  
 ", peak №2=" + peaks.get(1) +  
 ", peak №3=" + peaks.get(2) +  
 '}';  
 }  
  
 */\*\*  
 \* Проверка, находится ли точка внутри треугольника  
 \*  
 \** ***@param*** *point точка  
 \** ***@return*** *находится ли точка внутри треугольника  
 \*/* public boolean isInside(Vector2d point) {  
 return this.S == getSquareBy3Points(point, peaks.get(0), peaks.get(1)) +  
 getSquareBy3Points(point, peaks.get(0), peaks.get(2)) +  
 getSquareBy3Points(point, peaks.get(1), peaks.get(2));  
 }  
}

package app;  
  
import com.fasterxml.jackson.annotation.JsonCreator;  
import com.fasterxml.jackson.annotation.JsonProperty;  
import misc.Misc;  
import misc.Vector2d;  
  
import java.util.ArrayList;  
import java.util.List;  
  
*/\*\*  
 \* класс широкого луча  
 \*/*public class Beam {  
 */\*\*  
 \* Координаты вершин  
 \*/* public final ArrayList<Vector2d> peaks;  
  
 */\*\*  
 \* Конструктор широкого луча  
 \** ***@param*** *peaks вершины  
 \*/* @JsonCreator  
 public Beam(@JsonProperty("peaks") ArrayList<Vector2d> peaks) {  
 this.peaks = peaks;  
 }  
  
 */\*\*  
 \* Получить цвет широкого луча  
 \*  
 \** ***@return*** *цвет точки  
 \*/* public static int getColor() {  
 return Misc.*getColor*(0xCC, 0x00, 0xFF, 0x0);  
 }  
  
 */\*\*  
 \* Строковое представление объекта  
 \*  
 \** ***@return*** *строковое представление объекта  
 \*/* @Override  
 public String toString() {  
 return "Tringle{" +  
 "peak №1=" + peaks.get(0) +  
 ", peak №2=" + peaks.get(1) +  
 '}';  
 }  
  
 */\*\*  
 \* Проверка, находится ли точка внутри широкой полосы  
 \*  
 \** ***@param*** *point точка  
 \** ***@return*** *находится ли точка широкой полосы  
 \*/* public boolean isInside(Vector2d point) {  
 Vector2d l = Vector2d.*subtract*(peaks.get(1), peaks.get(0));  
 Vector2d d = Vector2d.*subtract*(point, peaks.get(0));  
 double projection = (l.x \* d.x + l.y \* d.y) / l.length();  
 return projection > 0 && projection < l.length() && (l.x \* d.y - l.y \* d.x > 0);  
 }  
}

# Приложение Б. UnitTest.java

import app.Beam;  
import app.Task;  
import app.Triangle;  
import misc.CoordinateSystem2d;  
import misc.Vector2d;  
import org.junit.Test;  
  
import java.util.ArrayList;  
import java.util.HashSet;  
import java.util.List;  
import java.util.Set;  
  
*/\*\*  
 \* Класс тестирования  
 \*/*public class UnitTest {  
 @Test  
 public void testTriangle () {  
 Triangle triangle = new Triangle(new ArrayList<>(List.*of*(new Vector2d(-2, -2), new Vector2d(0, 4), new Vector2d(2, -2))));  
 assert triangle.isInside(new Vector2d(0, 0));  
 assert !triangle.isInside(new Vector2d(5, 0));  
 assert !triangle.isInside(new Vector2d(-5, 0));  
 assert !triangle.isInside(new Vector2d(0, - 6));  
 }  
  
 @Test  
 public void testBeam() {  
 Beam beam = new Beam(new ArrayList<>(List.*of*(new Vector2d(0, 2), new Vector2d(2, 0))));  
 assert !beam.isInside(new Vector2d(1, 0));  
 assert !beam.isInside(new Vector2d(0, 0));  
 assert !beam.isInside(new Vector2d(-3, -3));  
 assert !beam.isInside(new Vector2d(5, 0));  
 assert !beam.isInside(new Vector2d(-5, 0));  
 assert beam.isInside(new Vector2d(6, 6));  
 assert beam.isInside(new Vector2d(6, 5));  
 }  
  
 @Test  
 public void testSolve() {  
 ArrayList<Triangle> triangles = new ArrayList<>();  
 triangles.add(new Triangle(new ArrayList<>(List.*of*(  
 new Vector2d(-7, -5),  
 new Vector2d(-1, 1),  
 new Vector2d(6, 5)))));  
 triangles.add(new Triangle(new ArrayList<>(List.*of*(  
 new Vector2d(-5, 7),  
 new Vector2d(-2, 4),  
 new Vector2d(4, 9)  
 ))));  
  
 ArrayList<Beam> beams = new ArrayList<>();  
 beams.add(new Beam(new ArrayList<>(List.*of*(  
 new Vector2d(7, -1),  
 new Vector2d(-5, 6)  
 ))));  
 beams.add(new Beam(new ArrayList<>(List.*of*(  
 new Vector2d(1, -4),  
 new Vector2d(9, -9)  
 ))));  
  
 Task task = new Task(new CoordinateSystem2d(-10, -10, 10, 10), triangles, beams);  
 task.solve();  
  
 assert task.getIndexBeam() == 0 && task.getIndexTriangle() == 0;  
 }  
}