Комитет по образованию г.Санкт-Петербург

ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБЩЕОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ПРЕЗИДЕНТСКИЙ ФИЗИКО-МАТЕМАТИЧЕСКИЙ

ЛИЦЕЙ №239

Отчет о практике

«Создание графических приложений на языке Java»

Учащийся 10-1 класса

Шонин Г.А.

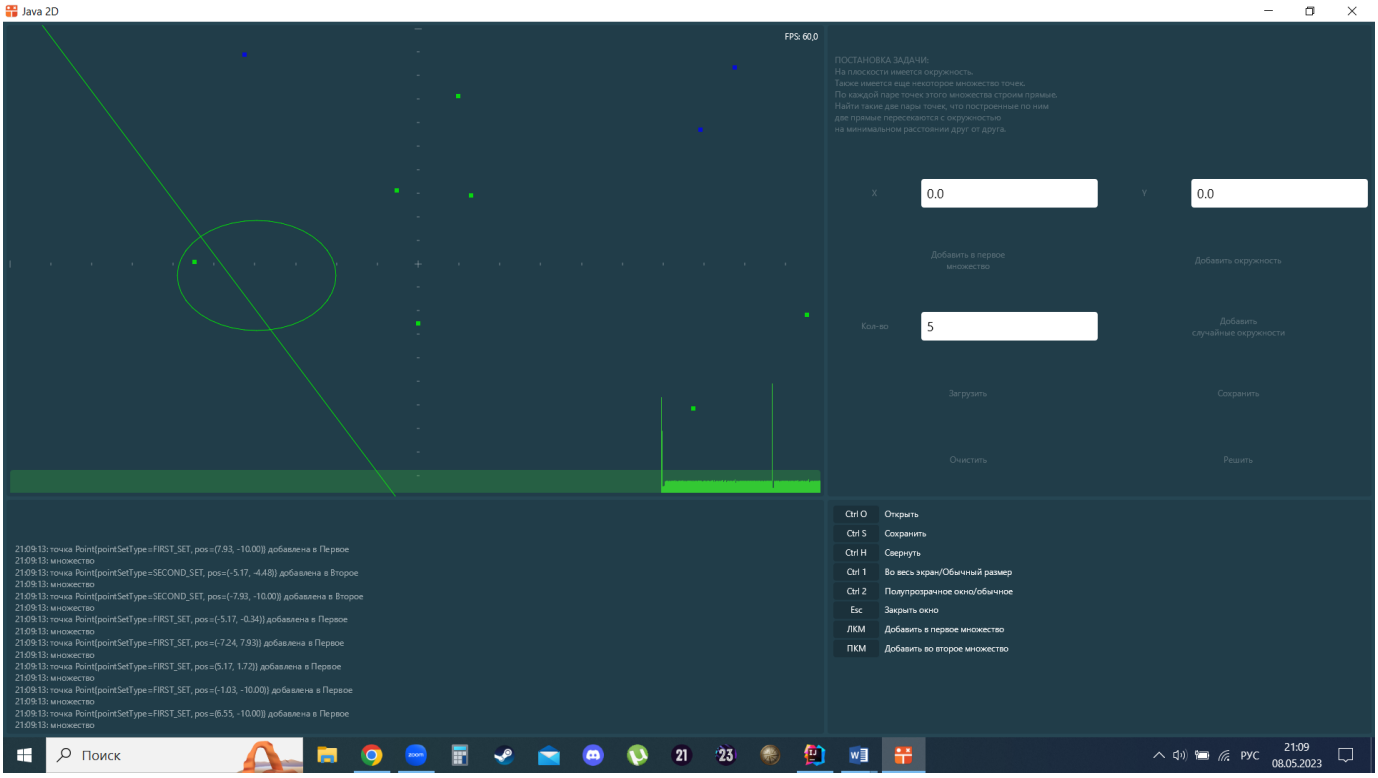
Преподаватель:

Клюнин А.О.

Санкт-Петербург - 2023 год

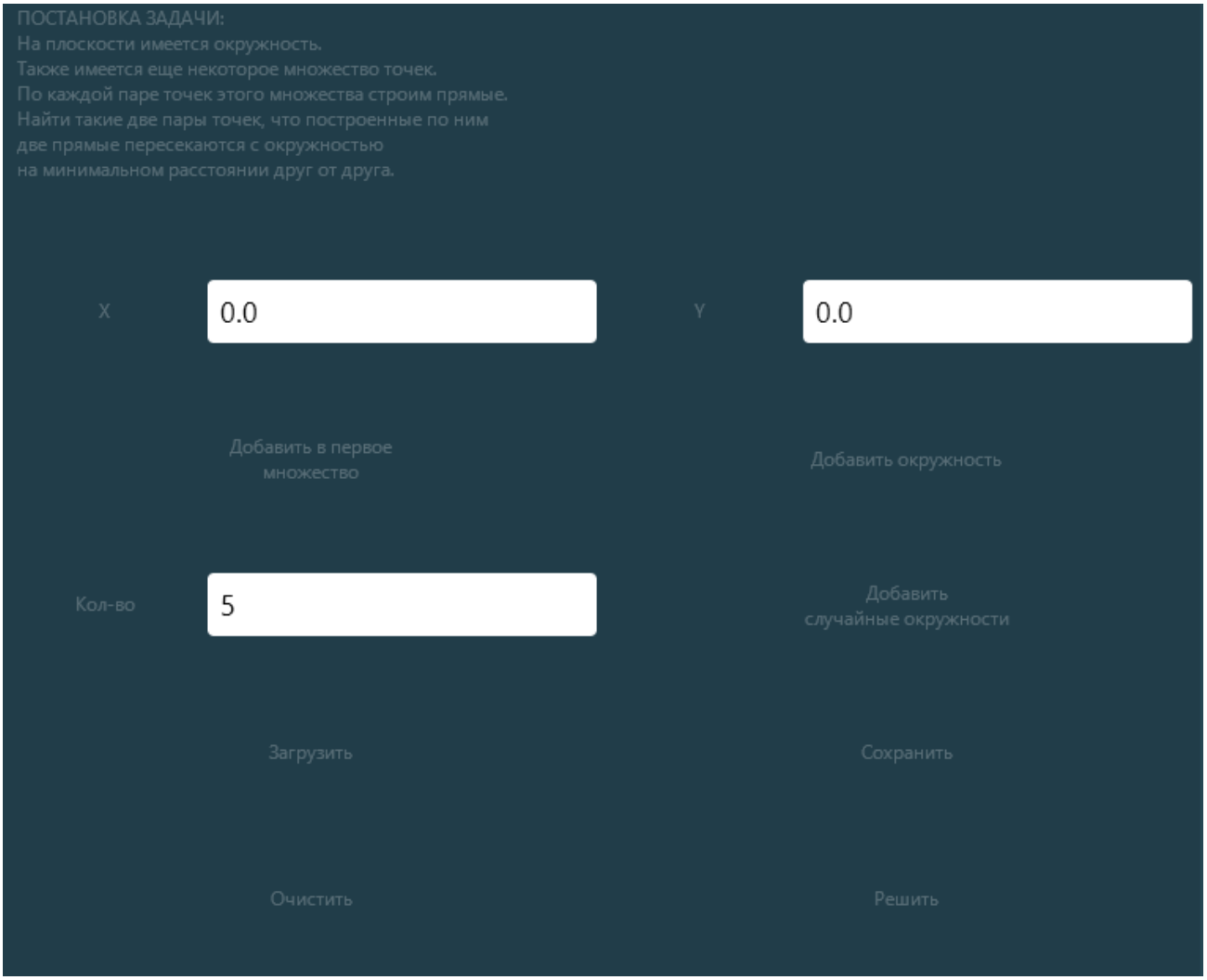
1.Постановка задачи

На плоскости имеется окружность. Также имеется еще некоторое множество точек. По каждой паре точек этого множества строим прямые. Найти такие две пары точек, что построенные по ним две прямые пересекаются с окружностью на минимальном расстоянии друг от друга. Выделить найденные точки, нарисовать прямые, построенные по ним. Выделить точки пересечения этих прямых с окружностью. Выделить дугу между этими точками пересечения. Дополнительная опция: искомые две пары точек не должны иметь общих точек.



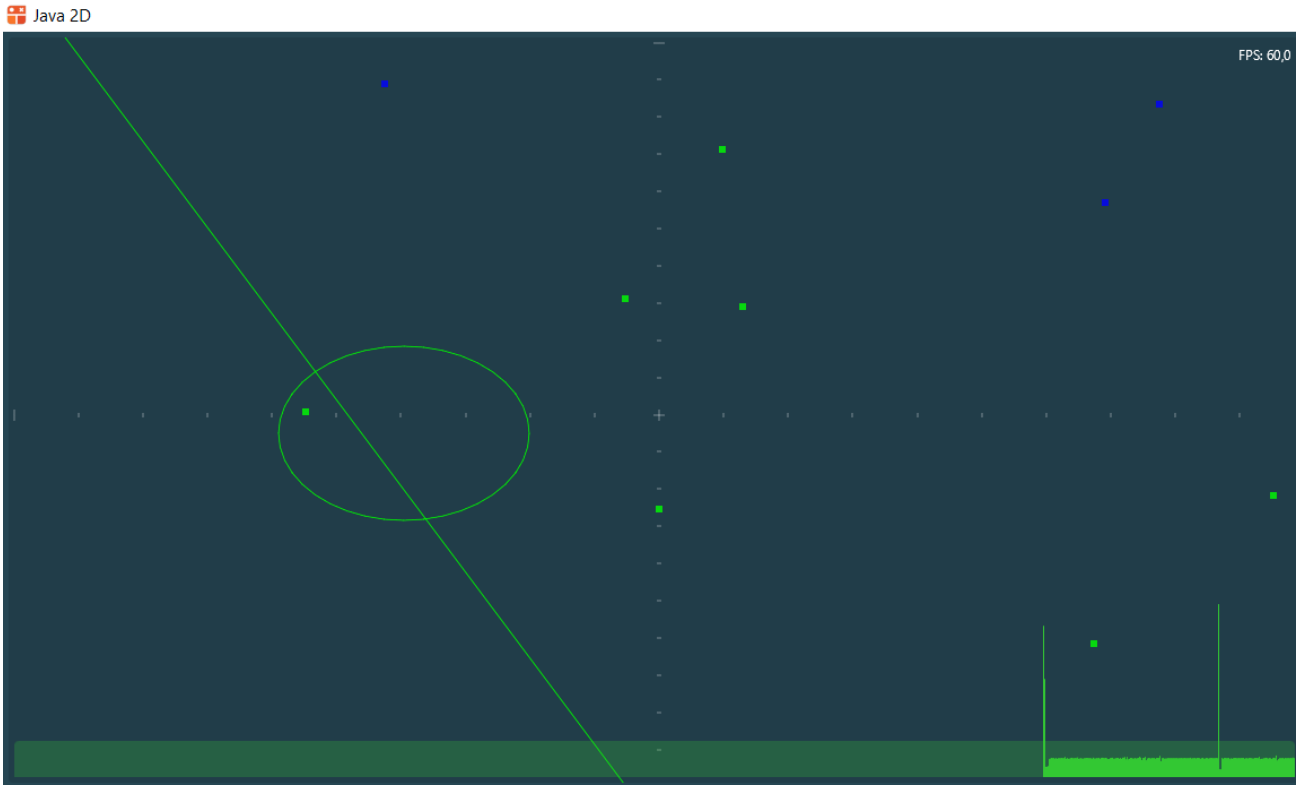
2.Элементы управления

В рамках данной задачи нужно было реализовать множество элементов управления. Они показаны ниже:



Для добавления точки по координатам было создано два поля ввода: «X» и «Y». Для добавления окружности использовалась функция от четырех координат(координат центра и координаты точки на окружности, по ним считался радиус). Также были созданы примитивы прямых и окружностей, для построения прямых по двум точкам был написан специальный метод.

Кроме того, написанная программа способна добавлять случайные точки, принадлежащие разным множествам по нажатию кнопок мыши. Аналогичные варианты действий написаны для добавления случайной окружности. Все это можно наблюдать в области рисования.



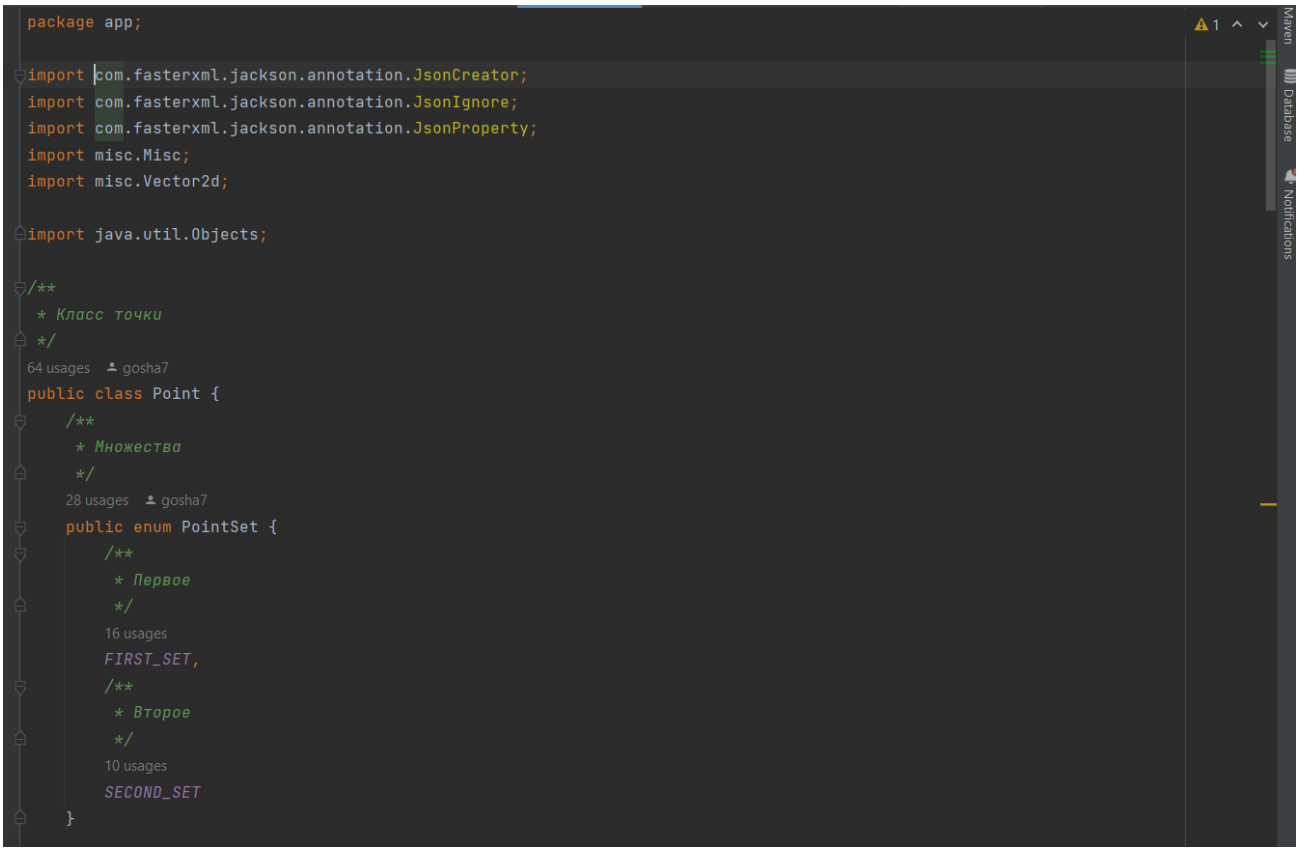
По нажатию левой или правой кнопки мыши будут добавлять случайные точки в области рисования. Так, при нажатии на левую кнопку мыши точка добавляется в 1 множество(зеленый цвет), а при нажатии на правую—во второе множество(синий цвет). Аналогично при нажатии кнопки Shift также будут добавлять окружности разных цветов.

3.Структуры данных

Для того чтобы хранить точки, был разработан класс **Point.java**. Аналогичные классы были разработаны для хранения окружностей и прямых.

В него были добавлены поля **pos**, соответствующее положению точки в пространстве задачи и тип множества **pointset**. Хранение типа множества обеспечено за счёт введения нового перечисления **PointSet**.

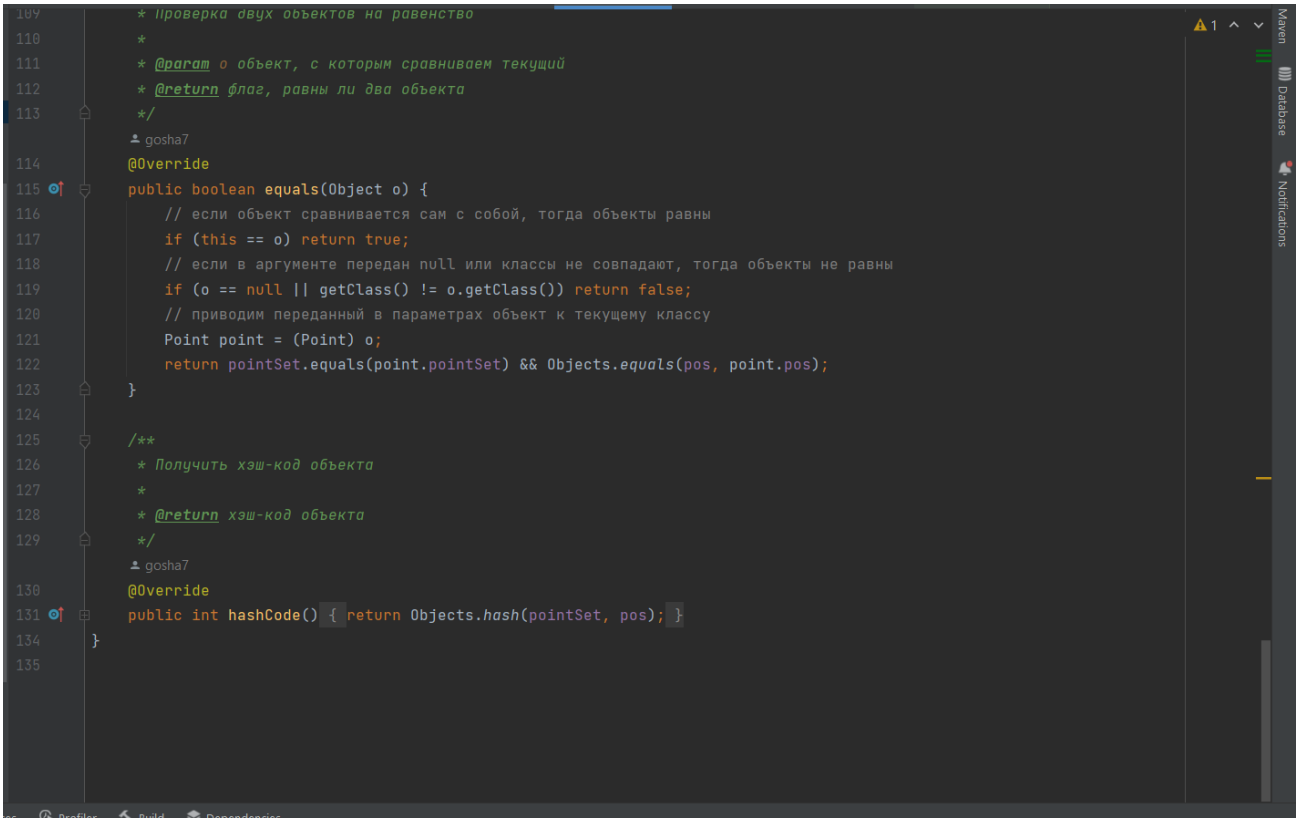
В случае с окружностями и прямыми ситуация была несколько более сложной:нужно было хранить и задавать объекты по нескольким координатам, а также создавать большее количество типов множеств. Пример класса и методов для него на примере **Point.java** представлен ниже.



```java
}

/**
 * Множество, которому принадлежит точка
 */
10 usages
protected final PointSet pointSet;
/**
 * Координаты точки
 */
10 usages
public final Vector2d pos;

/**
 * Конструктор точки
 *
 * @param pos     положение точки
 * @param setType множество, которому она принадлежит
 */
gosha7
@JsonCreator
public Point(@JsonProperty("pos") Vector2d pos, @JsonProperty("setType") PointSet setType) {
    this.pos = pos;
    this.pointSet = setType;
}

/**
 * Получить цвет точки по её множеству
 *
 * @return цвет точки
 */
```

```java
 */
1 usage  gosha7
@JsonIgnore
public int getColor() {
    return switch (pointSet) {
        case FIRST_SET -> Misc.getColor( a: 0xCC,  r: 0x00,  g: 0x00,  b: 0xFF);
        case SECOND_SET -> Misc.getColor( a: 0xCC,  r: 0x00,  g: 0xFF,  b: 0x0);
    };
}

/**
 * Получить положение
 * (нужен для json)
 *
 * @return положение
 */
2 usages  gosha7
public Vector2d getPos() { return pos; }

/**
 * Получить множество
 *
 * @return множество
 */
no usages  gosha7
public PointSet getSetType() { return pointSet; }

/**
 * Получить название множества
 *
 * @return название множества
 */
```

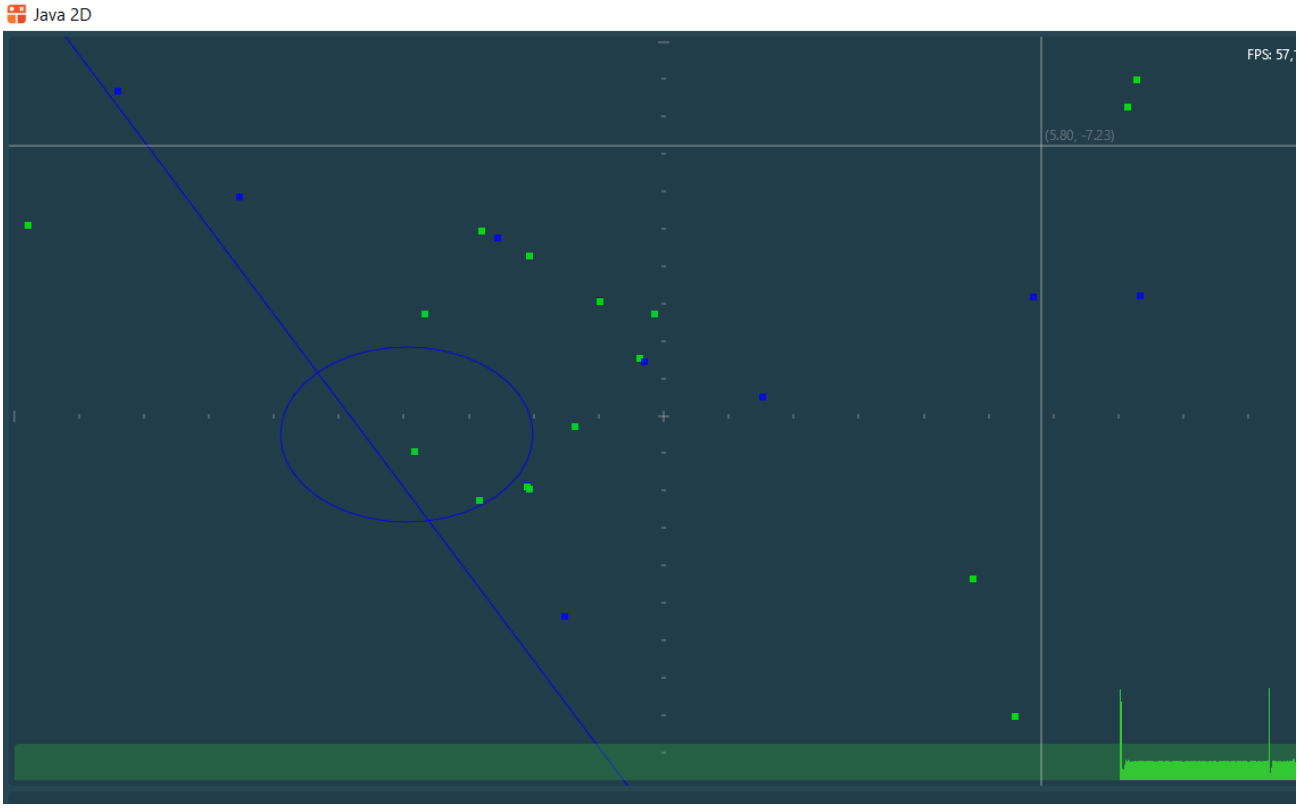
```
87      @JsonIgnore
88      public String getSetName() {
89          return switch (pointSet) {
90              case FIRST_SET -> "Первое множество";
91              case SECOND_SET -> "Второе множество";
92          };
93      }
94
95      /**
96       * Строковое представление объекта
97       *
98       * @return строковое представление объекта
99       */
        gosha7
100     @Override
101     public String toString() {
102         return "Point{" +
103                 "pointSetType=" + pointSet +
104                 ", pos=" + pos +
105                 '}';
106     }
107
108     /**
109      * Проверка двух объектов на равенство
110      *
111      * @param o объект, с которым сравниваем текущий
112      * @return флаг, равны ли два объекта
113      */
        gosha7
114     @Override
115     public boolean equals(Object o) {
```



```
109      * Проверка двух объектов на равенство
110      *
111      * @param o объект, с которым сравниваем текущий
112      * @return флаг, равны ли два объекта
113      */
        gosha7
114     @Override
115     public boolean equals(Object o) {
116         // если объект сравнивается сам с собой, тогда объекты равны
117         if (this == o) return true;
118         // если в аргументе передан null или классы не совпадают, тогда объекты не равны
119         if (o == null || getClass() != o.getClass()) return false;
120         // приводим переданный в параметрах объект к текущему классу
121         Point point = (Point) o;
122         return pointSet.equals(point.pointSet) && Objects.equals(pos, point.pos);
123     }
124
125     /**
126      * Получить хэш-код объекта
127      *
128      * @return хэш-код объекта
129      */
        gosha7
130     @Override
131     public int hashCode() { return Objects.hash(pointSet, pos); }
134 }
135
```

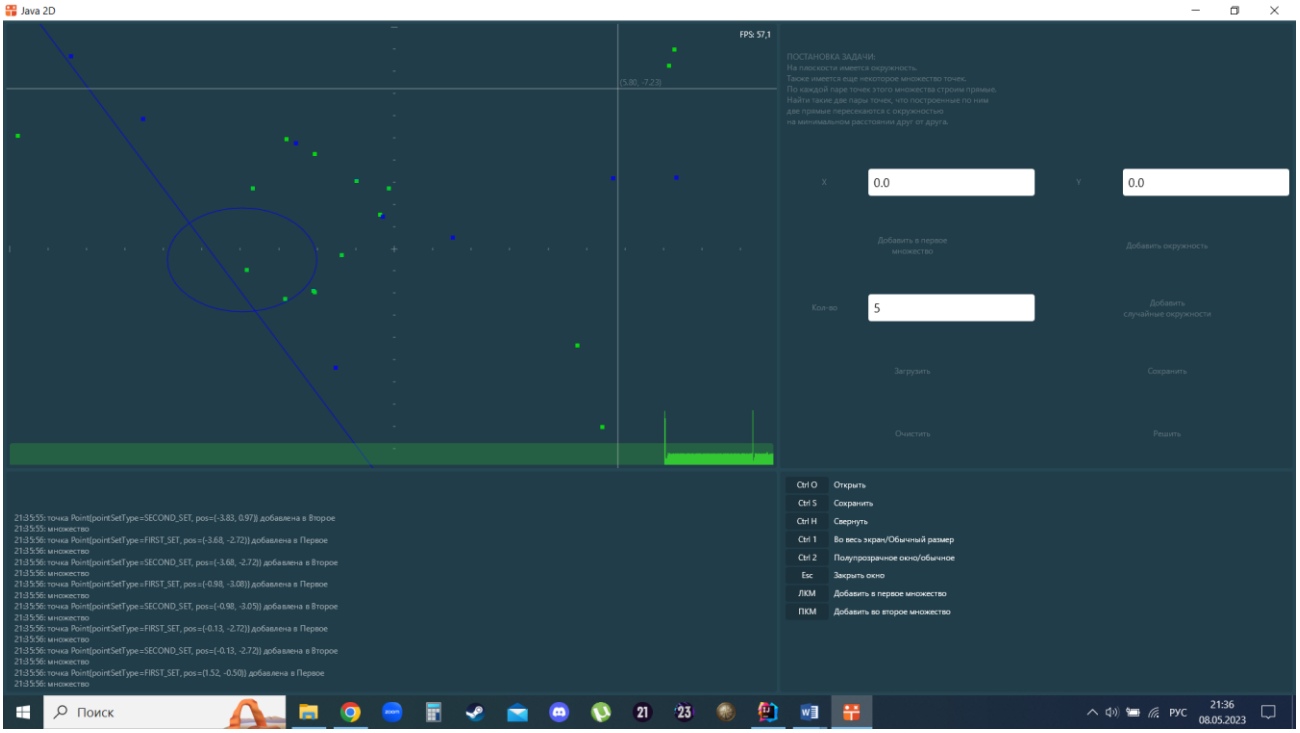
4.Рисование

Чтобы нарисовать точку, использовалась команда рисования прямоугольников **canvas.drawRect().** Аналогичные методы с более сложным методом были применены для рисования окружности и прямой. Ниже будет показана область рисования:



5.Решение задачи

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



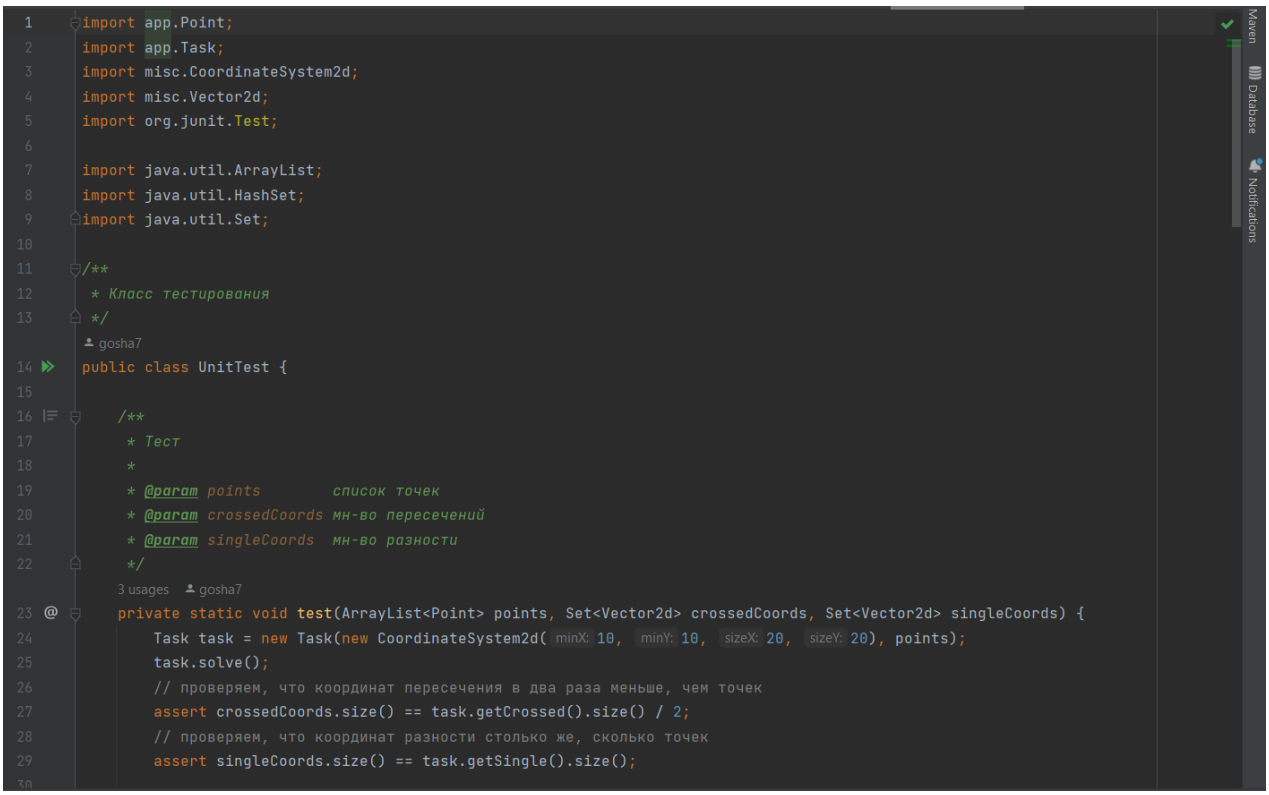
Программа проводит через все точки прямые и с помощью метода поиска координаты пересечения окружности и прямой находит для каждой пары точек нужные координаты. Далее по полученным координатам находит каждое расстояние между этими точками и ищет минимальное, после этого отмечает эти прямые и точки.

Программа сама решает задачу при нажатии на кнопку решить. Кроме того, можно сохранять файлы с решением задачи, а также загружать нужные файлы с помощью библиотеки json.

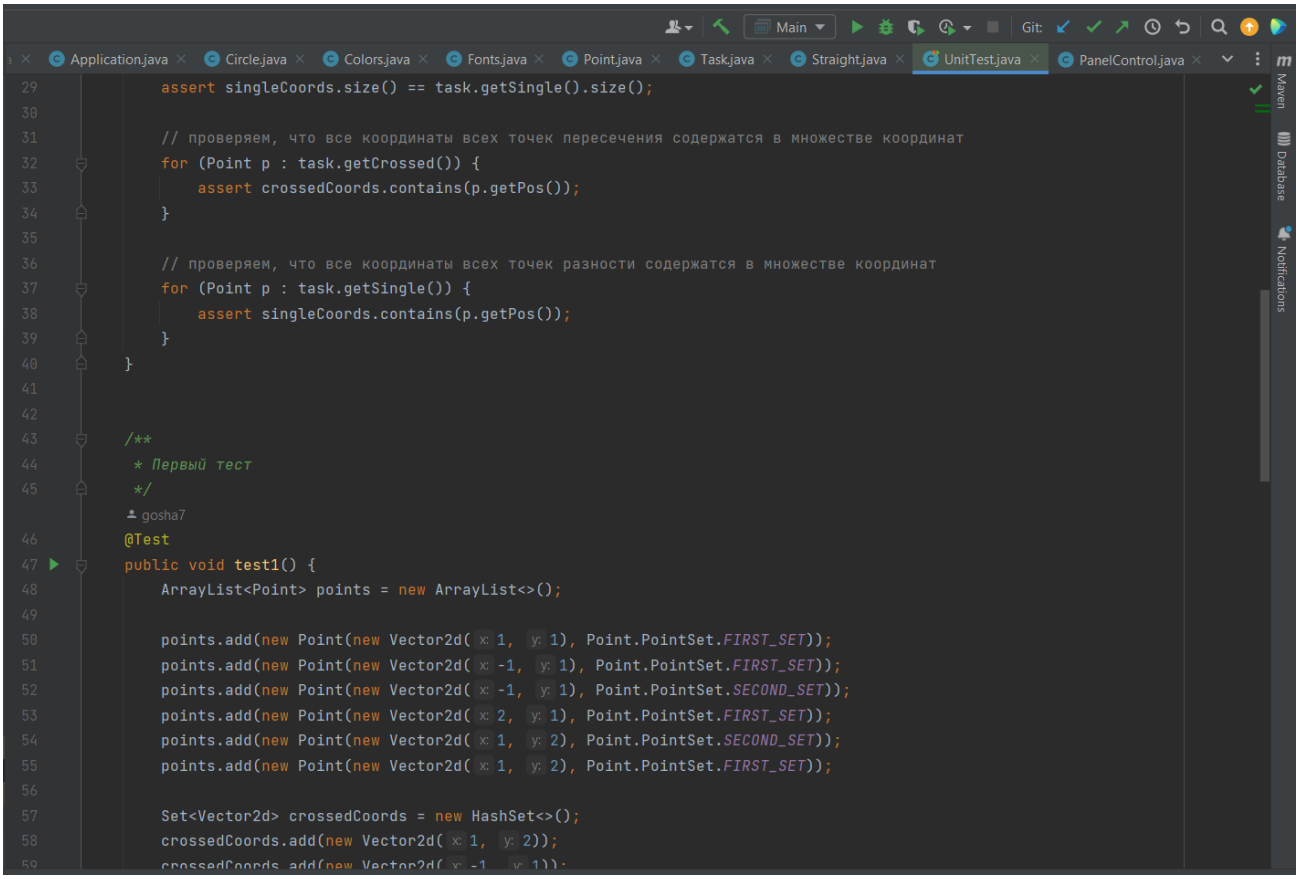
6.Проверка

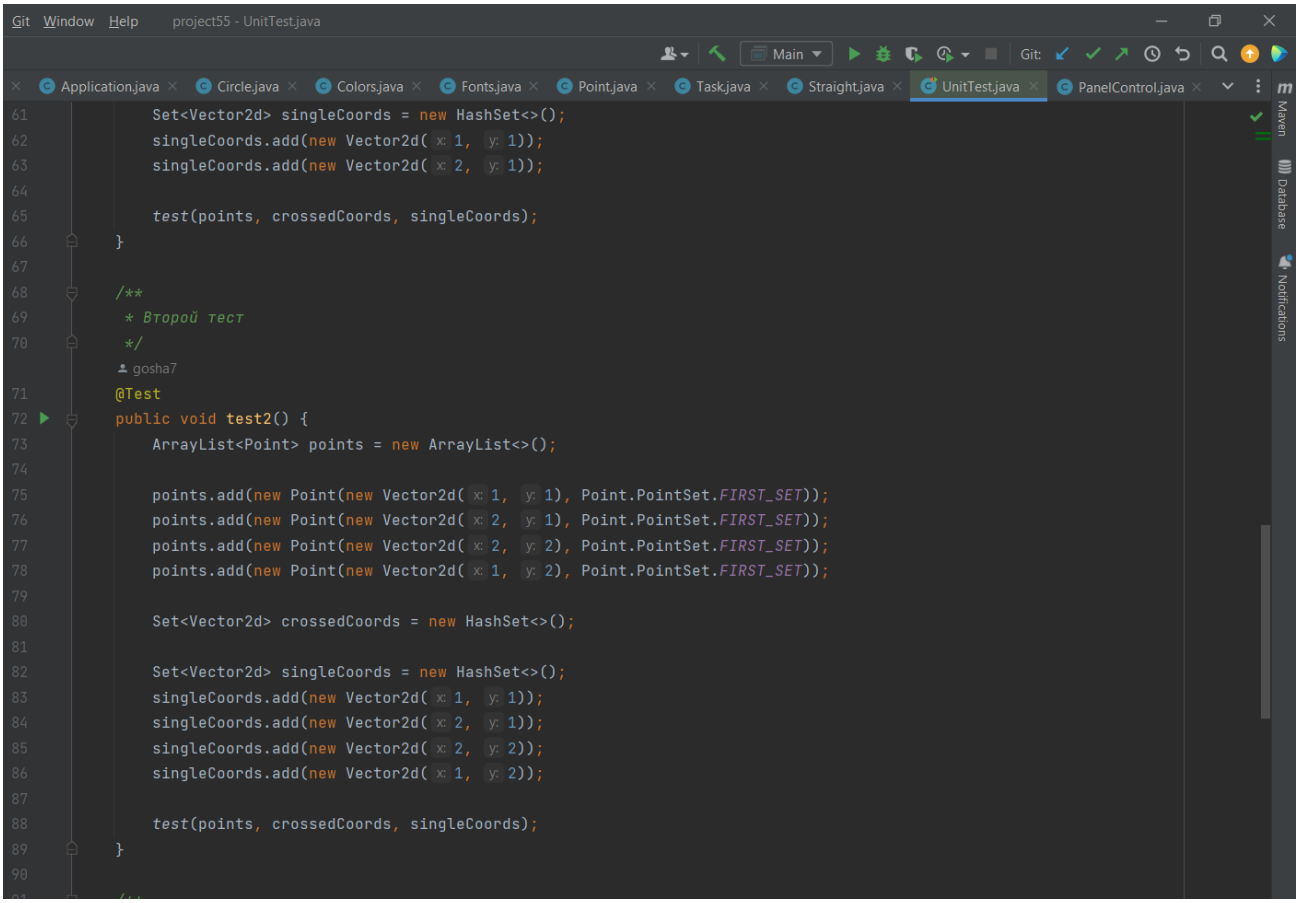
Для проверки правильности решенной задачи были разработаны unit-тесты.

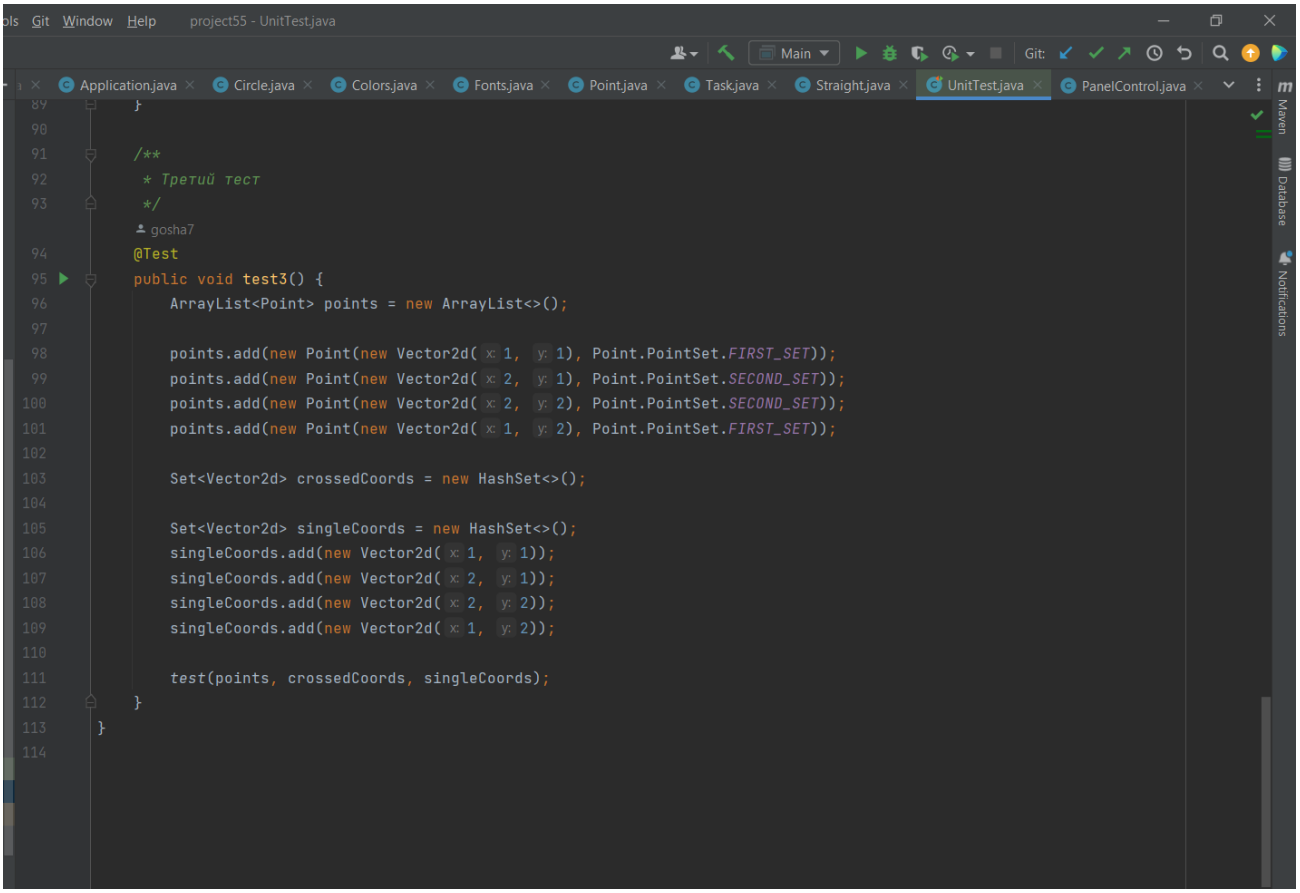
Приведем код этих тестов. Программа верно решила поставленную задачу.



```java
import app.Point;
import app.Task;
import misc.CoordinateSystem2d;
import misc.Vector2d;
import org.junit.Test;

import java.util.ArrayList;
import java.util.HashSet;
import java.util.Set;

/**
 * Класс тестирования
 */
gosha7
public class UnitTest {

    /**
     * Тест
     *
     * @param points        список точек
     * @param crossedCoords мн-во пересечений
     * @param singleCoords  мн-во разности
     */
    3 usages  gosha7
    private static void test(ArrayList<Point> points, Set<Vector2d> crossedCoords, Set<Vector2d> singleCoords) {
        Task task = new Task(new CoordinateSystem2d( minX: 10,  minY: 10,  sizeX: 20,  sizeY: 20), points);
        task.solve();
        // проверяем, что координат пересечения в два раза меньше, чем точек
        assert crossedCoords.size() == task.getCrossed().size() / 2;
        // проверяем, что координат разности столько же, сколько точек
        assert singleCoords.size() == task.getSingle().size();
```


Main
Git:
Application.java
Circle.java
Colors.java
Fonts.java
Point.java
Task.java
Straight.java
UnitTest.java
PanelControl.java
Maven
Database
Notifications
assert singleCoords.size() == task.getSingle().size();
// проверяем, что все координаты всех точек пересечения содержатся в множестве координат
for (Point p : task.getCrossed()) {
assert crossedCoords.contains(p.getPos());
}
// проверяем, что все координаты всех точек разности содержатся в множестве координат
for (Point p : task.getSingle()) {
assert singleCoords.contains(p.getPos());
}
}
/**
* Первый тест
*/
gosha7
@Test
public void test1() {
ArrayList<Point> points = new ArrayList<>();
points.add(new Point(new Vector2d(x: 1, y: 1), Point.PointSet.FIRST_SET));
points.add(new Point(new Vector2d(x: -1, y: 1), Point.PointSet.FIRST_SET));
points.add(new Point(new Vector2d(x: -1, y: 1), Point.PointSet.SECOND_SET));
points.add(new Point(new Vector2d(x: 2, y: 1), Point.PointSet.FIRST_SET));
points.add(new Point(new Vector2d(x: 1, y: 2), Point.PointSet.SECOND_SET));
points.add(new Point(new Vector2d(x: 1, y: 2), Point.PointSet.FIRST_SET));
Set<Vector2d> crossedCoords = new HashSet<>();
crossedCoords.add(new Vector2d(x: 1, y: 2));


Git Window Help project55 - UnitTest.java
Main
Git:
Application.java
Circle.java
Colors.java
Fonts.java
Point.java
Task.java
Straight.java
UnitTest.java
PanelControl.java
Maven
Database
Notifications
Set<Vector2d> singleCoords = new HashSet<>();
singleCoords.add(new Vector2d(x: 1, y: 1));
singleCoords.add(new Vector2d(x: 2, y: 1));
test(points, crossedCoords, singleCoords);
}
/**
* Второй тест
*/
gosha7
@Test
public void test2() {
ArrayList<Point> points = new ArrayList<>();
points.add(new Point(new Vector2d(x: 1, y: 1), Point.PointSet.FIRST_SET));
points.add(new Point(new Vector2d(x: 2, y: 1), Point.PointSet.FIRST_SET));
points.add(new Point(new Vector2d(x: 2, y: 2), Point.PointSet.FIRST_SET));
points.add(new Point(new Vector2d(x: 1, y: 2), Point.PointSet.FIRST_SET));
Set<Vector2d> crossedCoords = new HashSet<>();
Set<Vector2d> singleCoords = new HashSet<>();
singleCoords.add(new Vector2d(x: 1, y: 1));
singleCoords.add(new Vector2d(x: 2, y: 1));
singleCoords.add(new Vector2d(x: 2, y: 2));
singleCoords.add(new Vector2d(x: 1, y: 2));
test(points, crossedCoords, singleCoords);
}


ols Git Window Help project55 - UnitTest.java
Main
Git:
Application.java
Circle.java
Colors.java
Fonts.java
Point.java
Task.java
Straight.java
UnitTest.java
PanelControl.java
Maven
Database
Notifications
}
/**
* Третий тест
*/
gosha7
@Test
public void test3() {
ArrayList<Point> points = new ArrayList<>();
points.add(new Point(new Vector2d(x: 1, y: 1), Point.PointSet.FIRST_SET));
points.add(new Point(new Vector2d(x: 2, y: 1), Point.PointSet.SECOND_SET));
points.add(new Point(new Vector2d(x: 2, y: 2), Point.PointSet.SECOND_SET));
points.add(new Point(new Vector2d(x: 1, y: 2), Point.PointSet.FIRST_SET));
Set<Vector2d> crossedCoords = new HashSet<>();
Set<Vector2d> singleCoords = new HashSet<>();
singleCoords.add(new Vector2d(x: 1, y: 1));
singleCoords.add(new Vector2d(x: 2, y: 1));
singleCoords.add(new Vector2d(x: 2, y: 2));
singleCoords.add(new Vector2d(x: 1, y: 2));
test(points, crossedCoords, singleCoords);
}
}

7.Заключение

В рамках выполнения поставленной задачи было создано графическое приложение с требуемым функционалом. Правильность решения задачи проверена с помощью юнит-тестов. Были изучены множество функций и методов, примененные в ходе создания программы.