Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ

ОТЧЁТ

по дисциплине «Программирование на Java»

Обучающийся: Малахаева Валерия Ивановна гр. 09–132

(ФИО студента) (Группа)

Казань – 2023

Оглавление

[Часть 1. Консольное приложение 5](#_Toc154503202)

[**Цель** 5](#_Toc154503203)

[**Класс Point** 6](#_Toc154503204)

[**Класс Point2D** 8](#_Toc154503205)

[**Класс Point3D** 9](#_Toc154503206)

[**Класс Circle** 11](#_Toc154503207)

[**Класс NGon** 12](#_Toc154503208)

[**Класс Polyline** 14](#_Toc154503209)

[**Класс QGon** 15](#_Toc154503210)

[**Класс Rectangle** 16](#_Toc154503211)

[**Класс Segment** 18](#_Toc154503212)

[**Класс TGon** 19](#_Toc154503213)

[**Класс Trapeze** 20](#_Toc154503214)

[**Класс Figures** 21](#_Toc154503215)

[**Результат работы** 23](#_Toc154503216)

[Часть 2. Приложение с графическим интерфейсом на JavaFX 24](#_Toc154503217)

[**Цель** 24](#_Toc154503218)

[**Ход работы** 24](#_Toc154503219)

[**Контроллер AddFiguresController** 24](#_Toc154503220)

[**Контроллер CrossFiguresController** 26](#_Toc154503221)

[**Контроллер DeleteFiguresController** 27](#_Toc154503222)

[**Контроллер MainController** 28](#_Toc154503223)

[**Контроллер MoveFiguresController** 28](#_Toc154503224)

[**Контроллер PerimeterFigureController** 28](#_Toc154503225)

[**Контроллер SquareFigureController** 28](#_Toc154503226)

[**Класс FiguresApp** 29](#_Toc154503227)

[**Интерфейс** 29](#_Toc154503228)

[**Основная форма** 29](#_Toc154503229)

[**Пункт “Добавить фигуру”** 30](#_Toc154503230)

[**Пункт “Сохранить файл”** 31](#_Toc154503231)

[**Пункт “Загрузить файл”** 31](#_Toc154503232)

[**Пункт “Сохранить изображение”** 32](#_Toc154503233)

[**Пункт “Переместить фигуру”** 33](#_Toc154503234)

[**Пункт “Удалить фигуру”** 35](#_Toc154503235)

[**Пункт “Очистить”** 36](#_Toc154503236)

[**Пункт “Площадь фигуры”** 38](#_Toc154503237)

[**Пункт “Периметр фигуры”** 39](#_Toc154503238)

[**Пункт “Пересечение фигур”** 41](#_Toc154503239)

[Часть 3. Внедрение классов фигур с помощью .jar файла в JavaFX приложение. Тестирование классов 41](#_Toc154503240)

[**Цель** 41](#_Toc154503241)

[**Ход работы** 41](#_Toc154503242)

[**Результат** 43](#_Toc154503243)

[Часть 4. Проверка покрытия кода тестами. Проверка на уязвимости и баги. Внедрение аннотаций. Внедрение БД 43](#_Toc154503244)

[**Описание этапов** 43](#_Toc154503245)

[Листинг кода 48](#_Toc154503246)

[**Point** 48](#_Toc154503247)

[**Point2D** 53](#_Toc154503248)

[**Point3D** 54](#_Toc154503249)

[**Circle** 55](#_Toc154503250)

[**NGon** 57](#_Toc154503251)

[**Polyline** 60](#_Toc154503252)

[**Rectangle** 64](#_Toc154503253)

[**Segment** 66](#_Toc154503254)

[**TGon** 68](#_Toc154503255)

[**Trapeze** 69](#_Toc154503256)

[**Figures** 71](#_Toc154503257)

[**PointTest** 75](#_Toc154503258)

[**Point2DTest** 82](#_Toc154503259)

[**Point3DTest** 84](#_Toc154503260)

[**CircleTest** 85](#_Toc154503261)

[**NGonTest** 90](#_Toc154503262)

[**PolylineTest** 97](#_Toc154503263)

[**QGonTest** 103](#_Toc154503264)

[**RectangleTest** 105](#_Toc154503265)

[**SegmentTest** 108](#_Toc154503266)

[**TGonTest** 112](#_Toc154503267)

[**TrapezeTest** 115](#_Toc154503268)

[**IShape** 117](#_Toc154503269)

[**OpenFigure** 118](#_Toc154503270)

[**OpenFigureTest** 118](#_Toc154503271)

[**NullPointException** 119](#_Toc154503272)

[**AddFiguresController** 119](#_Toc154503273)

[**CrossFiguresController** 137](#_Toc154503274)

[**DeleteFiguresController** 142](#_Toc154503275)

[**DoubleArrayCodec** 146](#_Toc154503276)

[**FiguresApp** 147](#_Toc154503277)

[**MainController** 149](#_Toc154503278)

[**MoveFiguresController** 169](#_Toc154503279)

[**PerimeterController** 181](#_Toc154503280)

[**SquareFigureController** 184](#_Toc154503281)

[**Storage** 187](#_Toc154503282)

# Часть 1. Консольное приложение

## **Цель**

Создать консольное приложение, взаимодействующее с классами различных геометрических фигур. Реализовать данные классы и все необходимые методы. Подать на вход файлы с фигурами и типами взаимодействий, посмотреть результат.

## **Класс Point**

Поля:

- dim (размерность пространства) - защищенное поле типа int

- x (массив координат точки) - защищенное поле типа double[]

Конструкторы:

1. Конструктор с одним параметром:

- Point(int dim)

- Инициализирует поле dim переданным значением.

- Создает массив x размерности dim.

2. Конструктор с двумя параметрами:

- Point(int dim, double[] x)

- Проверяет соответствие размерности массива x заданной размерности пространства.

- Инициализирует поля dim и x переданными значениями.

Методы:

1. double getX(int i):

- Возвращает значение i-й координаты точки.

2. void setX(double[] x):

- Устанавливает новый массив координат точки.

- Проверяет соответствие размерности переданного массива x заданной размерности пространства.

3. void setX(double x, int i):

- Устанавливает новое значение i-й координаты точки.

- Проверяет правильность переданного индекса.

4. double abs():

- Вычисляет модуль точки (расстояние от начала координат).

5. static Point add(Point a, Point b):

- Возвращает новую точку, являющуюся суммой точек a и b.

- Проверяет совпадение размерностей точек.

6. Point add(Point b):

- Возвращает новую точку, являющуюся суммой текущей точки и точки b.

- Проверяет совпадение размерностей точек.

7. static Point sub(Point a, Point b):

- Возвращает новую точку, являющуюся разностью точек a и b.

- Проверяет совпадение размерностей точек.

8. Point sub(Point b):

- Возвращает новую точку, являющуюся разностью текущей точки и точки b.

- Проверяет совпадение размерностей точек.

9. static Point mult(Point a, double r):

- Возвращает новую точку, являющуюся результатом умножения координат точки a на число r.

10. Point mult(double r):

- Возвращает новую точку, являющуюся результатом умножения текущей точки на число r.

11. static double mult(Point a, Point b):

- Возвращает скалярное произведение точек a и b.

- Проверяет совпадение размерностей точек.

12. double mult(Point b):

- Возвращает скалярное произведение текущей точки и точки b.

- Проверяет совпадение размерностей точек.

13. static Point symAxis(Point a, int i):

- Возвращает новую точку, являющуюся симметричным отражением точки a относительно i-ой оси.

14. Point symAxis(int i):

- Возвращает новую точку, являющуюся симметричным отражением текущей точки относительно i-ой оси.

- Проверяет правильность номера оси.

15. Document toDocument():

- Возвращает объект Document, представляющий точку в виде документа.

- "type" - тип фигуры ("Point").

- "dim" - размерность пространства.

- "x" - массив координат точки.

## **Класс Point2D**

Конструкторы:

1. Конструктор без параметров:

- Point2D()

- Вызывает конструктор с параметром, устанавливая размерность пространства в 2.

2. Конструктор с параметром (массив координат):

- Point2D(double[] x)

- Вызывает конструктор с двумя параметрами, передавая размерность пространства и массив координат.

Методы:

1. static Point2D rot(Point2D a, double phi):

- Возвращает новую двумерную точку, полученную из данной точки a поворотом на угол phi.

2. Point2D rot(double phi):

- Возвращает новую двумерную точку, полученную из текущей точки поворотом на угол phi.

3. @Override

Point symAxis(int i):

- Возвращает новую двумерную точку, полученную из текущей точки симметрией относительно оси i.

- Переопределенный метод из базового класса.

- Проверяет правильность номера оси.

4. Document toDocument():

- Возвращает объект Document, представляющий двумерную точку в виде документа.

- "type" - тип фигуры ("Point2D").

- "dim" - размерность пространства.

- "x" - массив координат точки.

## **Класс Point3D**

Конструкторы:

1. Конструктор без параметров:

- Point3D()

- Вызывает конструктор с параметром, устанавливая размерность пространства в 3.

2. Конструктор с параметром (массив координат):

- Point3D(double[] x)

- Вызывает конструктор с двумя параметрами, передавая размерность пространства и массив координат.

Методы:

1. static Point3D cross\_prod(Point3D p1, Point3D p2):

- Возвращает новый объект типа "Point3D", представляющий векторное произведение двух векторов типа "Point3D".

2. Point3D cross\_prod(Point3D p):

- Возвращает новый объект типа "Point3D", представляющий векторное произведение текущего вектора и переданного в метод вектора типа "Point3D".

3. static double mix\_prod(Point3D p1, Point3D p2, Point3D p3):

- Возвращает значение смешанного произведения трех векторов типа "Point3D".

4. double mix\_prod(Point3D p1, Point3D p2):

- Возвращает значение смешанного произведения текущего вектора и двух переданных в метод векторов типа "Point3D".

5. Document toDocument():

- Возвращает объект Document, представляющий трехмерную точку в виде документа.

- "type" - тип фигуры ("Point3D").

- "dim" - размерность пространства.

- "x" - массив координат точки.

## **Класс Circle**

Поля:

- r (радиус окружности) - приватное поле типа double.

- p (координаты центра окружности, объект типа "Point2D") - приватное поле типа "Point2D".

Конструктор:

- Circle(Point2D p, double r):

- Принимает координаты центра окружности и ее радиус.

- Проверяет правильность введенных данных (радиус должен быть положительным числом).

- Присваивает значения полям.

Методы:

1. double square():

- Вычисляет и возвращает площадь окружности.

2. double length():

- Вычисляет и возвращает длину окружности.

3. IShape shift(Point2D a):

- Перемещает окружность на заданный вектор "a" и возвращает новый объект "Circle".

4. IShape rot(double phi):

- Поворачивает окружность на заданный угол "phi" и возвращает новый объект "Circle".

5. IShape symAxis(int i):

- Получает симметричную окружность относительно заданной оси "i" и возвращает новый объект "Circle".

6. boolean cross(IShape i):

- Проверяет пересечение данной окружности с другой фигурой.

- Входной параметр должен быть экземпляром класса "Circle".

- Вычисляет расстояние между центрами окружностей и проверяет условия пересечения.

7. Document toDocument():

- Возвращает объект "Document", представляющий окружность в виде документа.

- "type" - тип фигуры ("Circle").

- "r" - радиус окружности.

- "p" - координаты центра окружности в виде документа.

## **Класс NGon**

Поля:

- `n` (число вершин многоугольника) - защищенное поле типа `int`.

- `p` (массив точек вершин) - защищенное поле типа `Point2D[]`.

Конструктор:

- `NGon(Point2D[] p)`:

- Принимает массив точек вершин.

- Проверяет, что количество вершин не меньше трех, иначе выбрасывает исключение с сообщением об ошибке.

- Присваивает числу вершин значение длины массива точек вершин.

- Сохраняет массив точек вершин.

Методы:

1. `double square()`:

- Вычисляет и возвращает площадь многоугольника по формуле Гаусса через определитель матрицы.

2. `double length()`:

- Вычисляет и возвращает периметр многоугольника.

3. `IShape shift(Point2D a)`:

- Перемещает многоугольник на заданный вектор `a` и возвращает новый объект `NGon`.

4. `IShape rot(double phi)`:

- Поворачивает многоугольник на заданный угол `phi` и возвращает новый объект `NGon`.

5. `IShape symAxis(int i)`:

- Получает симметричный многоугольник относительно заданной оси симметрии `i` и возвращает новый объект `NGon`.

6. `boolean cross(IShape i)`:

- Проверяет пересечение данного многоугольника с другой фигурой.

- Входной параметр должен быть экземпляром класса `NGon`.

- Проверяет пересечение отрезков между всеми вершинами текущего многоугольника и другого.

7. `Document toDocument()`:

- Возвращает объект `Document`, представляющий многоугольник в виде документа.

- "type" - тип фигуры ("NGon").

- "n" - число вершин многоугольника.

- "points" - массив точек вершин многоугольника в виде документов.

## **Класс Polyline**

Поля:

- `n` (число точек ломаной линии) - приватное поле типа `int`.

- `p` (массив точек ломаной линии) - приватное поле типа `Point2D[]`.

Конструктор:

- `Polyline(Point2D[] p)`:

- Принимает массив точек ломаной линии.

- Присваивает числу точек значение длины массива точек ломаной линии.

- Сохраняет массив точек ломаной линии.

Методы:

1. `Point2D getP(int i)`:

- Возвращает точку ломаной линии по индексу `i`.

2. `void setP(Point2D p, int i)`:

- Задает точку ломаной линии по индексу `i`.

3. `double square()`:

- Возвращает 0, так как ломаная линия не имеет площади.

4. `double length()`:

- Вычисляет и возвращает длину ломаной линии.

5. `IShape shift(Point2D a)`:

- Перемещает ломаную линию на заданный вектор `a` и возвращает новый объект `Polyline`.

6. `IShape rot(double phi)`:

- Поворачивает ломаную линию на заданный угол `phi` и возвращает новый объект `Polyline`.

7. `IShape symAxis(int i)`:

- Получает симметричную ломаную линию относительно заданной оси симметрии `i` и возвращает новый объект `Polyline`.

8. `boolean cross(IShape i)`:

- Проверяет пересечение данной ломаной линии с другой фигурой.

- Входной параметр должен быть экземпляром класса `Polyline`.

- Проверяет пересечение отрезков между всеми вершинами текущей ломаной линии и другой.

9. `Document toDocument()`:

- Возвращает объект `Document`, представляющий ломаную линию в виде документа.

- "type" - тип фигуры ("Polyline").

- "n" - число точек ломаной линии.

- "points" - массив точек ломаной линии в виде документов.

## **Класс QGon**

Поля:

- Нет дополнительных полей, использует поля класса "NGon".

Конструктор:

- `QGon(Point2D[] p)`:

- Вызывает конструктор суперкласса "NGon", передавая ему массив точек.

Методы:

1. `double square()` (переопределенный метод):

- Вычисляет и возвращает площадь четырехугольника.

- Вычисляет длины сторон четырехугольника и использует формулу Герона для расчета площадей двух треугольников.

- Возвращает сумму площадей двух треугольников.

2. `IShape shift(Point2D a)` (переопределенный метод):

- Перемещает четырехугольник на заданный вектор `a` и возвращает новый объект "QGon".

3. `IShape rot(double phi)` (переопределенный метод):

- Поворачивает четырехугольник на заданный угол `phi` и возвращает новый объект "QGon".

4. `IShape symAxis(int i)` (переопределенный метод):

- Получает симметричный четырехугольник относительно заданной оси симметрии `i` и возвращает новый объект "QGon".

5. `Document toDocument()` (переопределенный метод):

- Возвращает объект "Document", представляющий четырехугольник в виде документа.

- "type" - тип фигуры ("QGon").

- "n" - число вершин многоугольника.

- "points" - массив точек многоугольника в виде документов.

## **Класс Rectangle**

Поля:

- Нет дополнительных полей, использует поля класса "QGon".

Конструктор:

- `Rectangle(Point2D[] p)`:

- Вызывает конструктор суперкласса "QGon", передавая ему массив точек прямоугольника.

- Производит дополнительную проверку на то, что входные точки действительно образуют прямоугольник. (Закомментированный код в конструкторе)

Методы:

1. `double square()` (переопределенный метод):

- Вычисляет и возвращает площадь прямоугольника по координатам его вершин.

2. `IShape shift(Point2D a)` (переопределенный метод):

- Перемещает прямоугольник на заданный вектор `a` и возвращает новый объект "Rectangle".

3. `IShape rot(double phi)` (переопределенный метод):

- Поворачивает прямоугольник на заданный угол `phi` и возвращает новый объект "Rectangle".

4. `IShape symAxis(int i)` (переопределенный метод):

- Получает симметричный прямоугольник относительно заданной оси симметрии `i` и возвращает новый объект "Rectangle".

5. `Document toDocument()` (переопределенный метод):

- Возвращает объект "Document", представляющий прямоугольник в виде документа.

- "type" - тип фигуры ("Rectangle").

- "n" - число вершин многоугольника.

- "points" - массив точек многоугольника в виде документов.

## **Класс Segment**

Поля:

- `start` (начальная точка сегмента)

- `finish` (конечная точка сегмента)

Конструктор:

- `Segment(Point2D s, Point2D f)`:

- Инициализирует начальную и конечную точки сегмента.

Методы:

1. `double length()` (переопределенный метод):

- Вычисляет и возвращает длину отрезка.

2. `IShape shift(Point2D a)` (переопределенный метод):

- Сдвигает отрезок на заданный вектор `a` и возвращает новый объект "Segment".

3. `IShape rot(double phi)` (переопределенный метод):

- Поворачивает отрезок на заданный угол `phi` и возвращает новый объект "Segment".

4. `IShape symAxis(int i)` (переопределенный метод):

- Получает симметричный отрезок относительно заданной оси симметрии `i` и возвращает новый объект "Segment".

5. `boolean cross(IShape i)` (переопределенный метод):

- Проверяет, пересекается ли отрезок с другой фигурой (в данном случае, с отрезком).

6. `Document toDocument()` (переопределенный метод):

- Возвращает объект "Document", представляющий отрезок в виде документа.

- "type" - тип фигуры ("Segment").

- "start" - начальная точка сегмента в виде документа.

- "finish" - конечная точка сегмента в виде документа.

7. `double square()` (переопределенный метод):

- Возвращает площадь отрезка. В данном случае, всегда возвращает 0, так как отрезок не имеет площади.

## **Класс TGon**

Конструктор:

- `TGon(Point2D[] p)`:

- Вызывает конструктор суперкласса "NGon", передавая ему массив точек `p`.

- Проверяет, что все три точки не лежат на одной прямой. В случае лежания на одной прямой выбрасывает `IllegalArgumentException`.

Методы:

1. `double square()` (переопределенный метод):

- Вычисляет и возвращает площадь треугольника по формуле Герона.

2. `IShape shift(Point2D a)` (переопределенный метод):

- Сдвигает треугольник на заданный вектор `a` и возвращает новый объект "TGon".

3. `IShape rot(double phi)` (переопределенный метод):

- Поворачивает треугольник на заданный угол `phi` и возвращает новый объект "TGon".

4. `IShape symAxis(int i)` (переопределенный метод):

- Получает симметричный треугольник относительно заданной оси симметрии `i` и возвращает новый объект "TGon".

5. `Document toDocument()` (переопределенный метод):

- Возвращает объект "Document", представляющий треугольник в виде документа.

- "type" - тип фигуры ("TGon").

- "n" - количество точек треугольника.

- "points" - массив документов, представляющих точки треугольника.

## **Класс Trapeze**

Конструктор:

- `Trapeze(Point2D[] p)`:

- Вызывает конструктор суперкласса "QGon", передавая ему массив точек `p`.

Методы:

1. `double square()` (переопределенный метод):

- Вычисляет и возвращает площадь трапеции по формуле: `0.5 \* a \* b \* abs(sin(phi))`, где:

- `a` - длина одной из параллельных сторон трапеции,

- `b` - длина другой параллельной стороны трапеции,

- `phi` - угол между параллельными сторонами.

2. `IShape shift(Point2D a)` (переопределенный метод):

- Сдвигает трапецию на заданный вектор `a` и возвращает новый объект "Trapeze".

3. `IShape rot(double phi)` (переопределенный метод):

- Поворачивает трапецию на заданный угол `phi` и возвращает новый объект "Trapeze".

4. `IShape symAxis(int i)` (переопределенный метод):

- Получает симметричную трапецию относительно заданной оси симметрии `i` и возвращает новый объект "Trapeze".

5. `Document toDocument()` (переопределенный метод):

- Возвращает объект "Document", представляющий трапецию в виде документа.

- "type" - тип фигуры ("Trapeze").

- "n" - количество точек трапеции.

- "points" - массив документов, представляющих точки трапеции.

## **Класс Figures**

Поля:

- `private final List<IShape> list`: Список фигур.

- `private final List<IShape> list1`: Второй список фигур.

- `private final List<IShape> list2`: Третий список фигур.

Конструкторы:

1. `public Figures()`:

- Устанавливает локаль по умолчанию.

- Вызывает метод `Figures()`.

2. `public void Figures()`:

- Метод для загрузки данных из файла, создания списков фигур, выполнения операций и вывода результатов.

Методы:

1. `private void Figures()`:

- Внутренний метод, часть конструктора, отвечающий за обработку данных из файла.

Примечания:

- При чтении данных из файла используется try-with-resources для автоматического закрытия ресурсов.

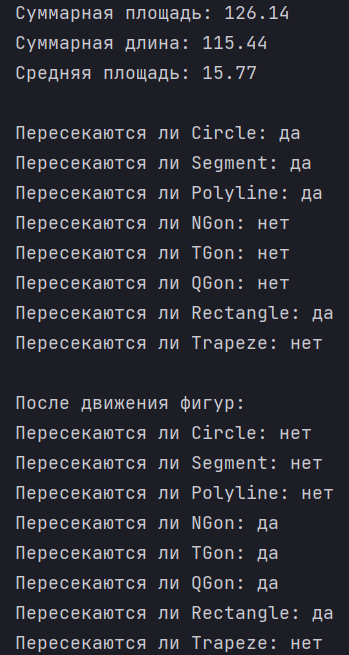
- Данные о фигурах считываются из файла "files/file.txt".

- Второй набор данных считывается из файла "files/file1.txt".

- Движения фигур задаются из файла "files/file2.txt".

- Результаты вычислений, включая площади, длины, среднюю площадь, и пересечения, выводятся на консоль.

## **Результат работы**



# Часть 2. Приложение с графическим интерфейсом на JavaFX

## **Цель**

Создать приложение на JavaFX, реализовать взаимодействие с классами при помощи графического интерфейса.

## **Ход работы**

* Создание отдельных сцен для реализации всего функционала.
* Создание контроллеров для обработки функционала каждой отдельной сцены, создание общего контроллера.
* Связывание объектов сцен с контроллерами.
* Реализация функционала.
* Настройка стилей.

## **Контроллер AddFiguresController**

Поля:

- `private MainController mainController`: Ссылка на основной контроллер приложения. Используется для взаимодействия с методами основного контроллера, такими как отрисовка фигур.

- `private CrossFiguresController crossFiguresController`: Ссылка на контроллер, отвечающий за обработку пересечений фигур.

- `private MoveFiguresController moveFiguresController`: Ссылка на контроллер, управляющий перемещением фигур.

- `private DeleteFiguresController deleteFiguresController`: Ссылка на контроллер, занимающийся удалением фигур.

- `private FiguresApp application`: Ссылка на основной класс приложения. Используется для доступа к методам приложения.

FXML-поля:

- `private Spinner<Integer> numericUpDown1`: Элемент интерфейса Spinner для выбора количества вершин фигуры.

- `private ComboBox<String> comboBox1`: Элемент интерфейса ComboBox для выбора типа фигуры.

- `private Pane pn1, pn2, ..., pn18`: Элементы интерфейса Pane для отображения вершин фигуры.

- `private Label label1`: Элемент интерфейса Label для отображения метки.

- `private TextField textBox1, textBox2, ..., textBox9`: Элементы интерфейса TextField для ввода координат.

Конструктор:

- `public AddFiguresController()`: Пустой конструктор.

Методы:

- `public void setControllers(MainController mainController, CrossFiguresController crossFiguresController, MoveFiguresController moveFiguresController, DeleteFiguresController deleteFiguresController)`: Метод, устанавливающий ссылки на контроллеры. Вызывается при инициализации приложения.

- `public void setApplication(FiguresApp application)`: Метод, устанавливающий ссылку на основной класс приложения. Вызывается при инициализации приложения.

- `public void initialize()`: Метод, вызываемый при инициализации контроллера, настраивает элементы интерфейса. Устанавливает слушатели событий.

- `public void comboBox1\_SelectedIndexChanged(ActionEvent actionEvent)`: Обработчик события выбора элемента в ComboBox. В зависимости от выбранного типа фигуры настраивает интерфейс.

- `public void button4\_Click(ActionEvent event)`: Обработчик события нажатия кнопки "Добавить". Проводит валидацию ввода, создает фигуру и добавляет ее в список фигур. Обновляет отображение на холсте.

- `public void update()`: Метод, обновляющий количество вершин фигуры в соответствии со значением Spinner.

- `public void buttonCancel\_Click(ActionEvent actionEvent)`: Обработчик события нажатия кнопки "Отмена". Закрывает текущее окно.

## **Контроллер CrossFiguresController**

Поля:

- `public static List<IShape> listFigures`: Статический список фигур для сравнения на пересечение.

- `public static List<String> names`: Статический список имен фигур.

- `@FXML public ComboBox<String> comboBox5`: Элемент интерфейса ComboBox для выбора первой фигуры для сравнения.

- `@FXML public ComboBox<String> comboBox6`: Элемент интерфейса ComboBox для выбора второй фигуры для сравнения.

- `@FXML public ComboBox<String> comboBox7`: Элемент интерфейса ComboBox для выбора типа фигуры для отображения.

Внешние ссылки:

- `private MainController mainController`: Ссылка на основной контроллер приложения.

- `private AddFiguresController addFiguresController`: Ссылка на контроллер для добавления фигур.

- `private MoveFiguresController moveFiguresController`: Ссылка на контроллер для перемещения фигур.

- `private DeleteFiguresController deleteFiguresController`: Ссылка на контроллер для удаления фигур.

- `private FiguresApp application`: Ссылка на основной класс приложения.

Методы:

- `public void setApplication(FiguresApp application)`: Устанавливает ссылку на основной класс приложения.

- `public void setControllers(MainController mainController, AddFiguresController addFiguresController, MoveFiguresController moveFiguresController, DeleteFiguresController deleteFiguresController)`: Устанавливает ссылки на контроллеры.

- `public void initialize()`: Метод, вызываемый при инициализации контроллера, заполняет элементы интерфейса данными.

- `@FXML public void comboBox7\_move(ActionEvent actionEvent)`: Обработчик события выбора типа фигуры для отображения. Заполняет ComboBox'ы списками фигур выбранного типа.

- `@FXML private void button7\_Click(ActionEvent actionEvent)`: Обработчик события нажатия кнопки "Пересечение". Выполняет проверку на пересечение выбранных фигур и выводит результат на холст.

- `public void CancelButton\_Click(ActionEvent actionEvent)`: Обработчик события нажатия кнопки "Отмена". Закрывает текущее окно.

## **Контроллер DeleteFiguresController**

Поля:

- `@FXML public ComboBox<String> comboBox4`: Элемент интерфейса ComboBox для выбора фигуры для удаления.

Внешние ссылки:

- `private MainController mainController`: Ссылка на основной контроллер приложения.

- `private AddFiguresController addFiguresController`: Ссылка на контроллер для добавления фигур.

- `private CrossFiguresController crossFiguresController`: Ссылка на контроллер для сравнения фигур на пересечение.

- `private MoveFiguresController moveFiguresController`: Ссылка на контроллер для перемещения фигур.

- `private FiguresApp application`: Ссылка на основной класс приложения.

Методы:

- `public void setApplication(FiguresApp application)`: Устанавливает ссылку на основной класс приложения.

- `public void setControllers(MainController mainController, AddFiguresController addFiguresController, CrossFiguresController crossFiguresController, MoveFiguresController moveFiguresController)`: Устанавливает ссылки на контроллеры.

- `public void initialize()`: Метод, вызываемый при инициализации контроллера, заполняет ComboBox фигурами для удаления.

- `@FXML private void button6\_Click(ActionEvent actionEvent)`: Обработчик события нажатия кнопки "Удалить". Удаляет выбранную фигуру и обновляет отображение на холсте.

- `public void CancelButton\_Click(ActionEvent actionEvent)`: Обработчик события нажатия кнопки "Отмена". Закрывает текущее окно.

## **Контроллер MainController**

Взаимодействие с основной сценой.

## **Контроллер MoveFiguresController**

Взаимодействие со сценой, отвечающей за перемещение фигур

## **Контроллер PerimeterFigureController**

Взаимодействие со сценой, отвечающей за вычисление периметра фигуры.

## **Контроллер SquareFigureController**

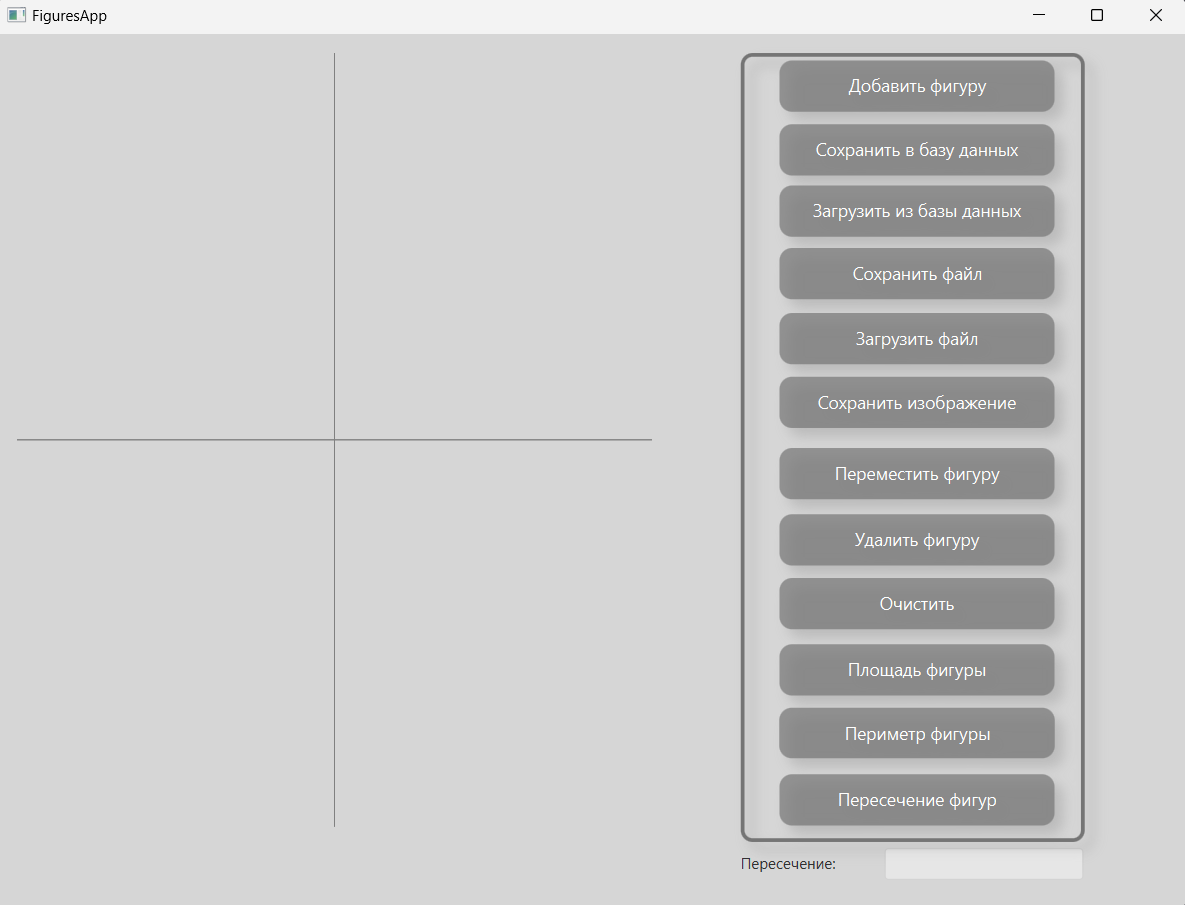
Взаимодействие со сценой, отвечающей за вычисление площади фигуры.

## **Класс FiguresApp**

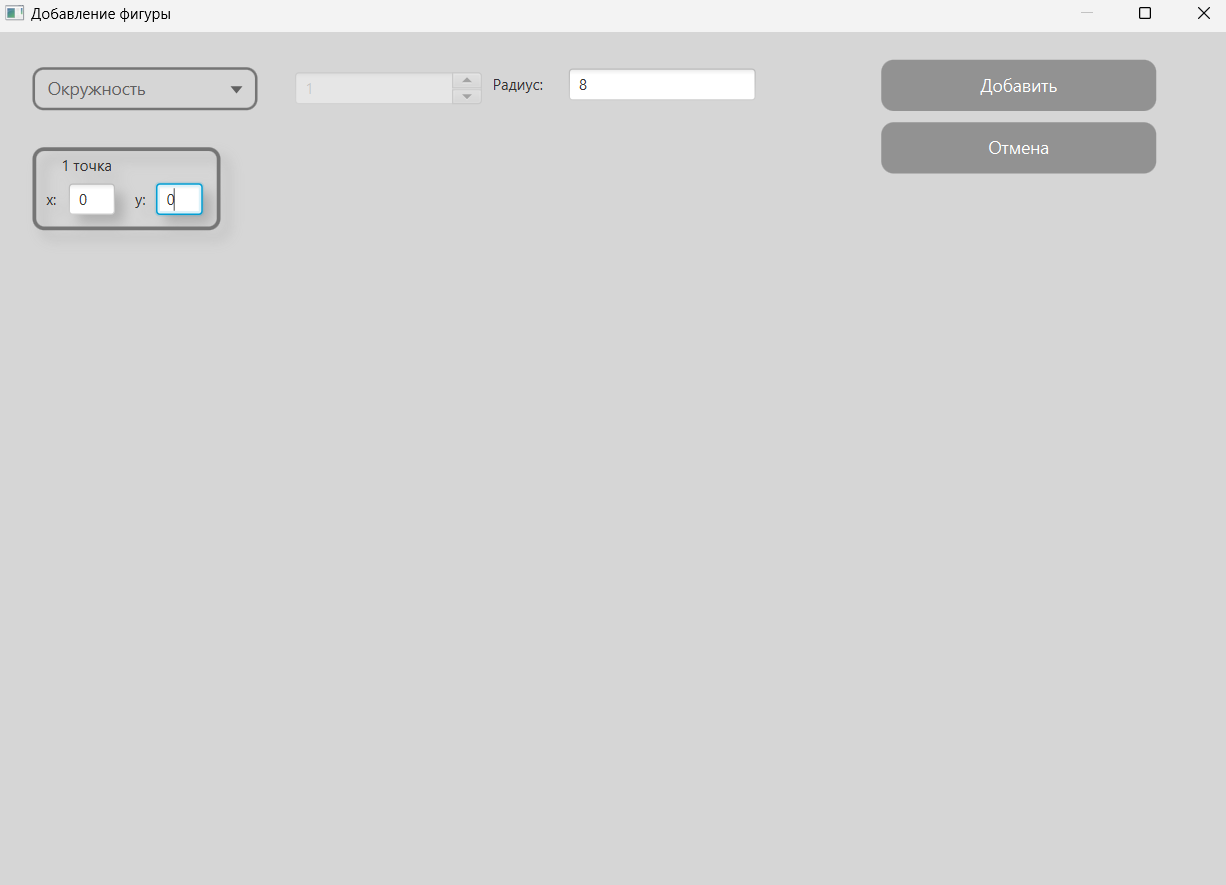
Устанавливает взаимодействие между различными контроллерами и сценами.

## **Интерфейс**

### **Основная форма**



### **Пункт “Добавить фигуру”**

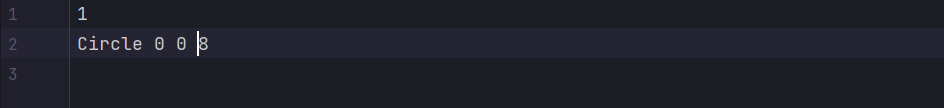


Результат



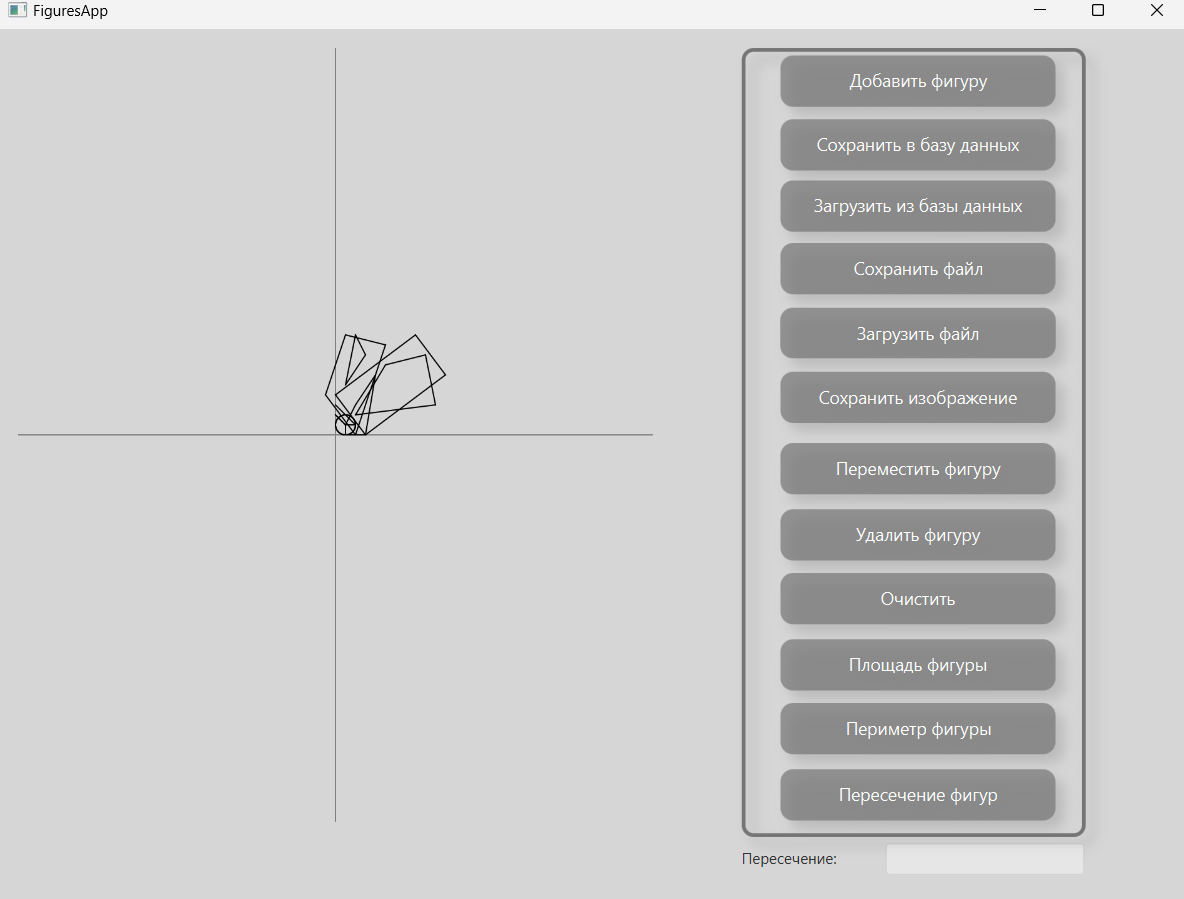
### **Пункт “Сохранить файл”**

Результат



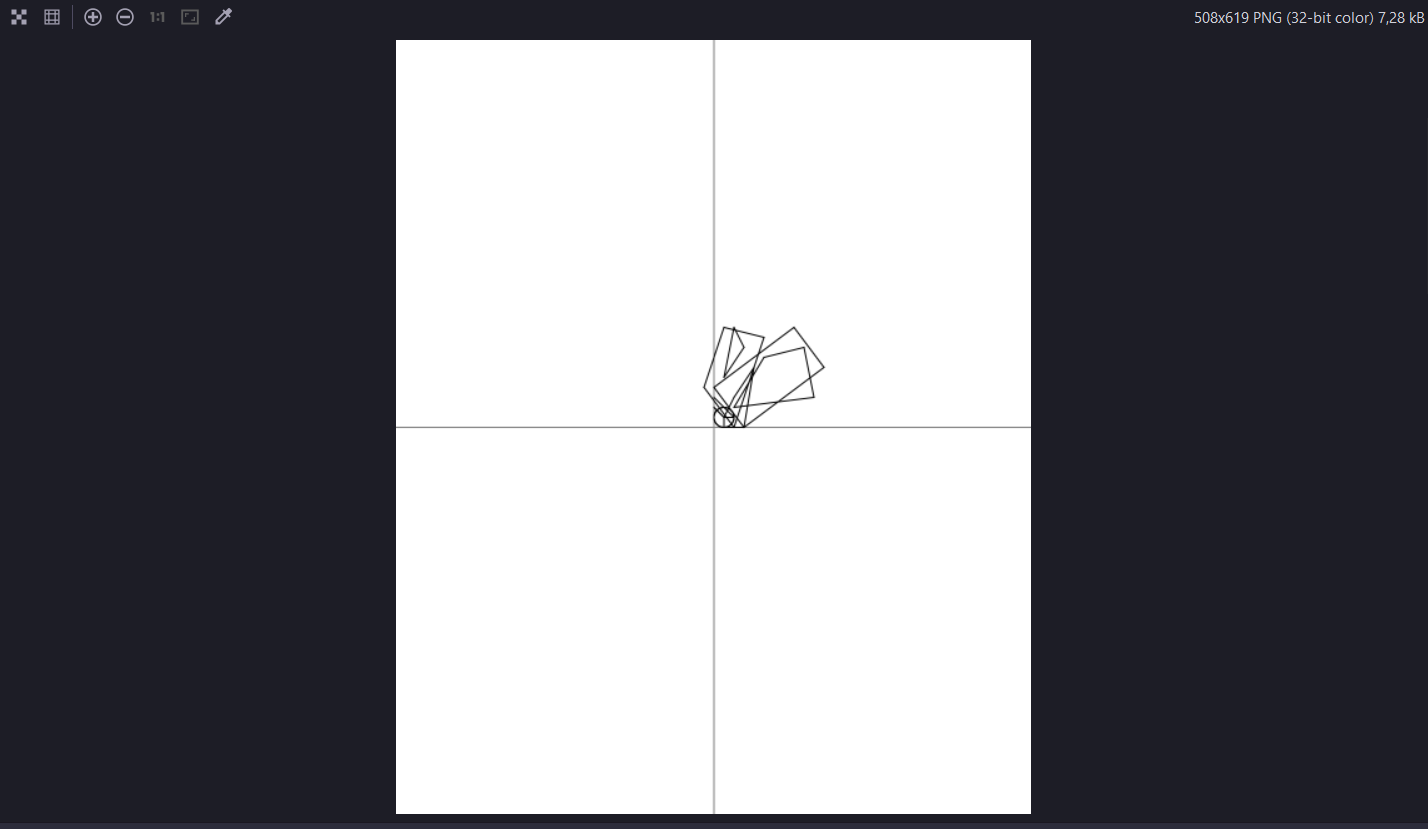
### **Пункт “Загрузить файл”**

Результат

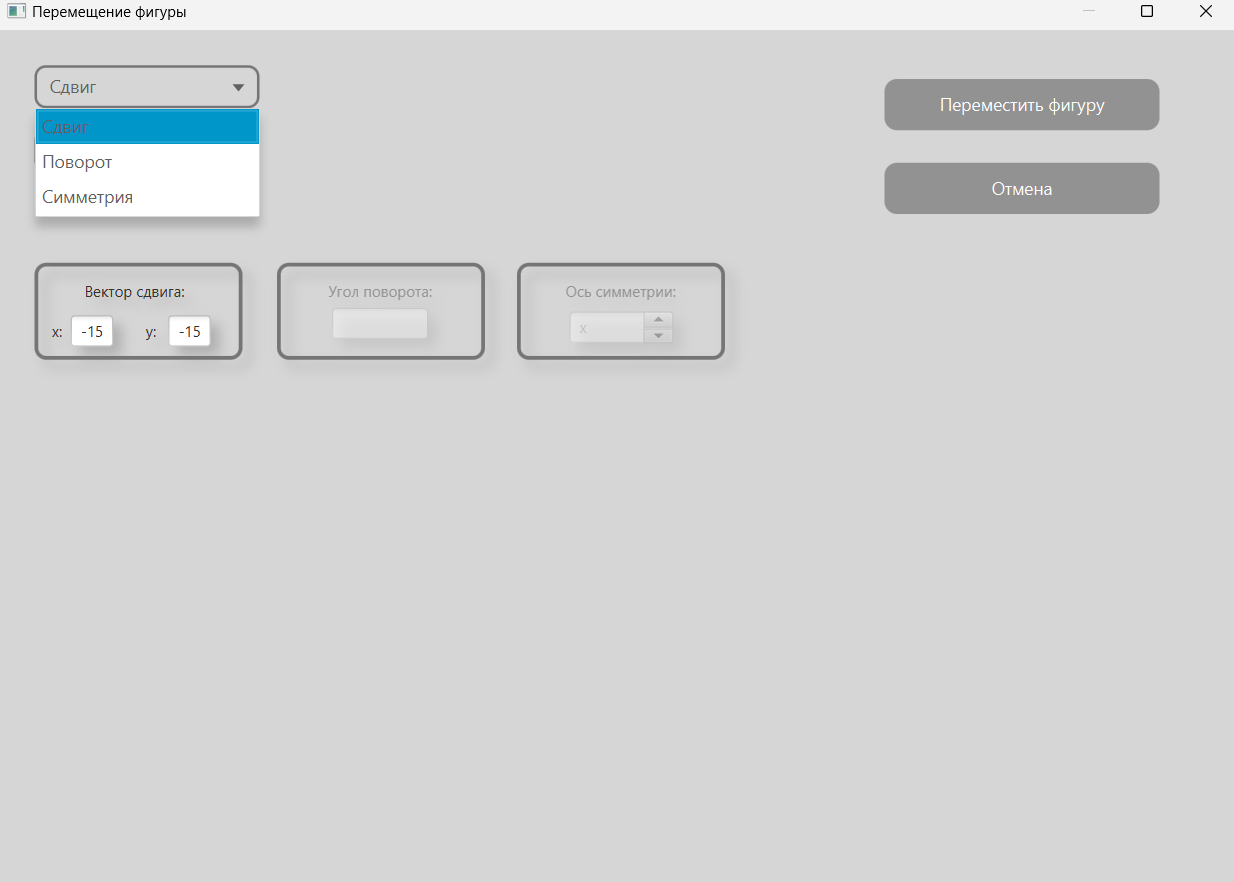


### **Пункт “Сохранить изображение”**

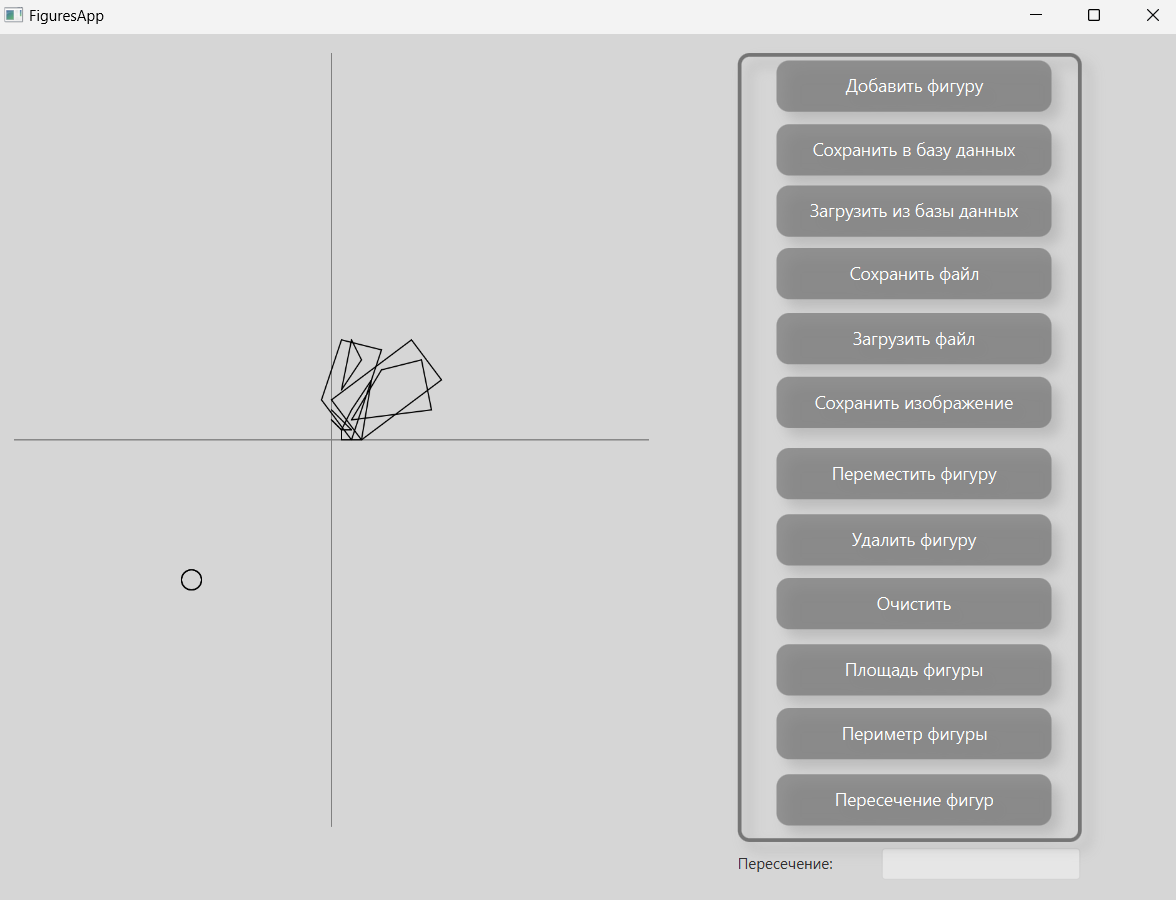
Результат



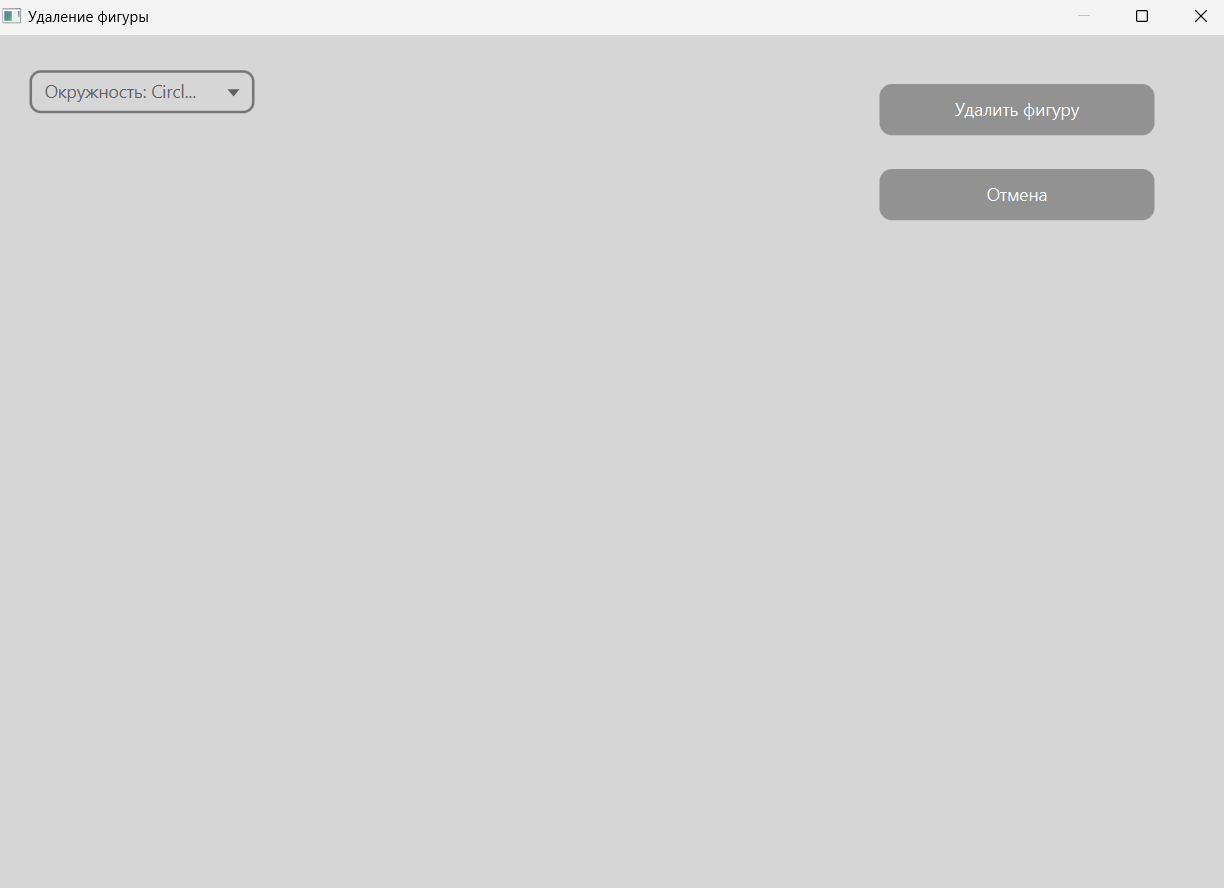
### **Пункт “Переместить фигуру”**



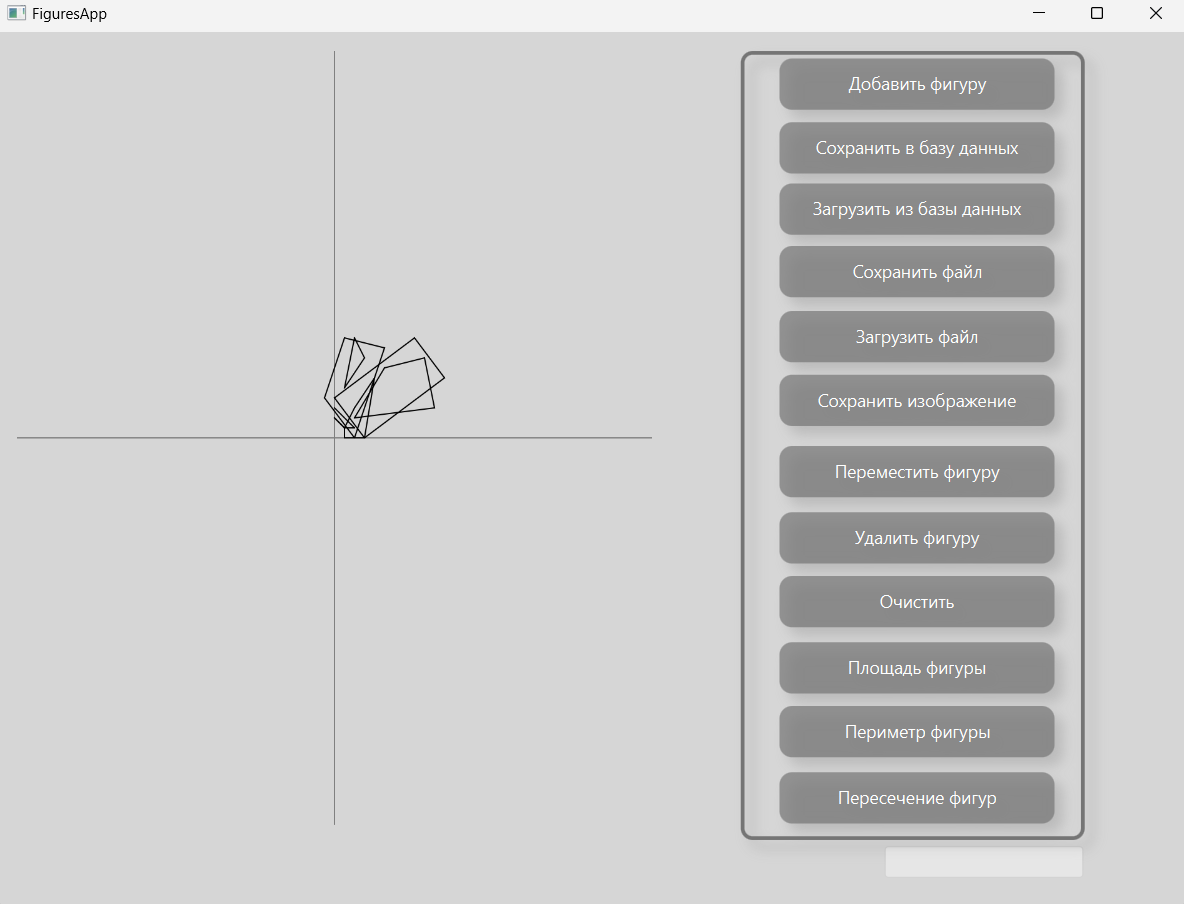
Результат



### **Пункт “Удалить фигуру”**

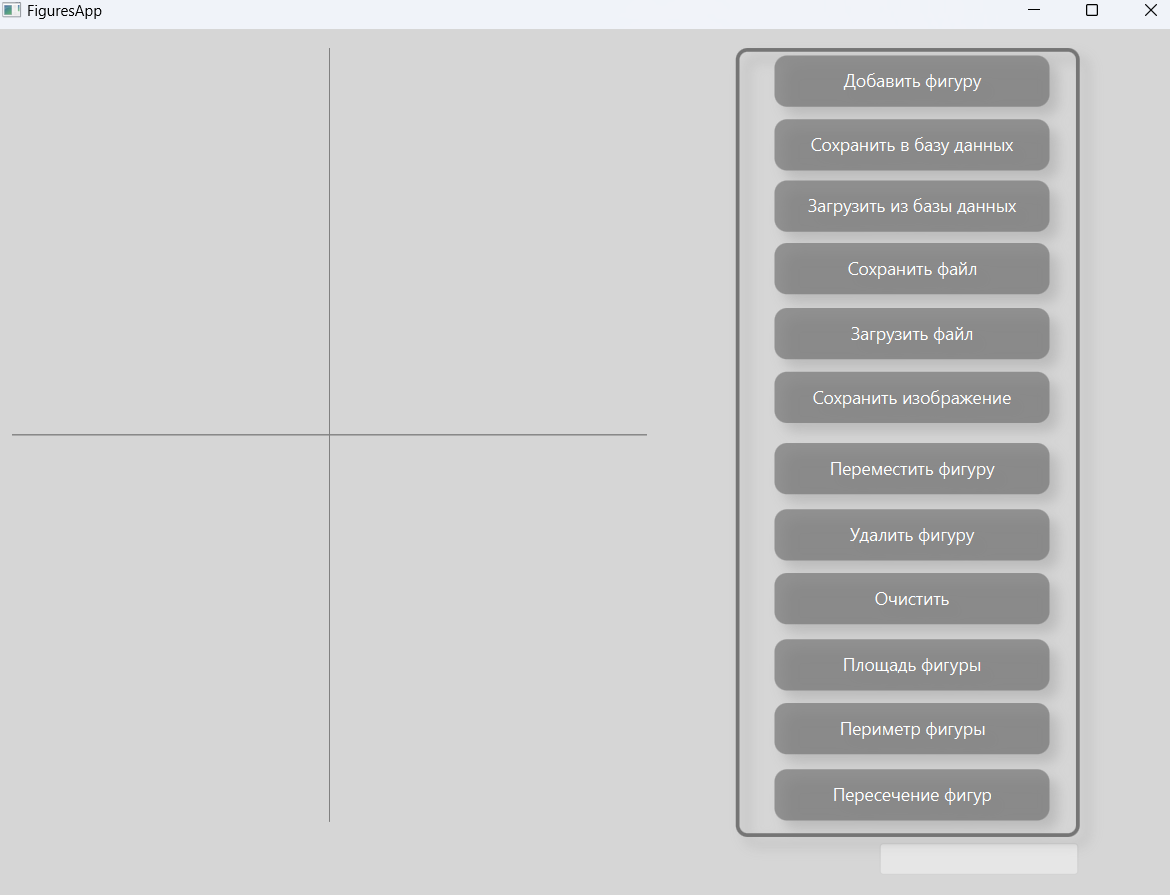


Результат

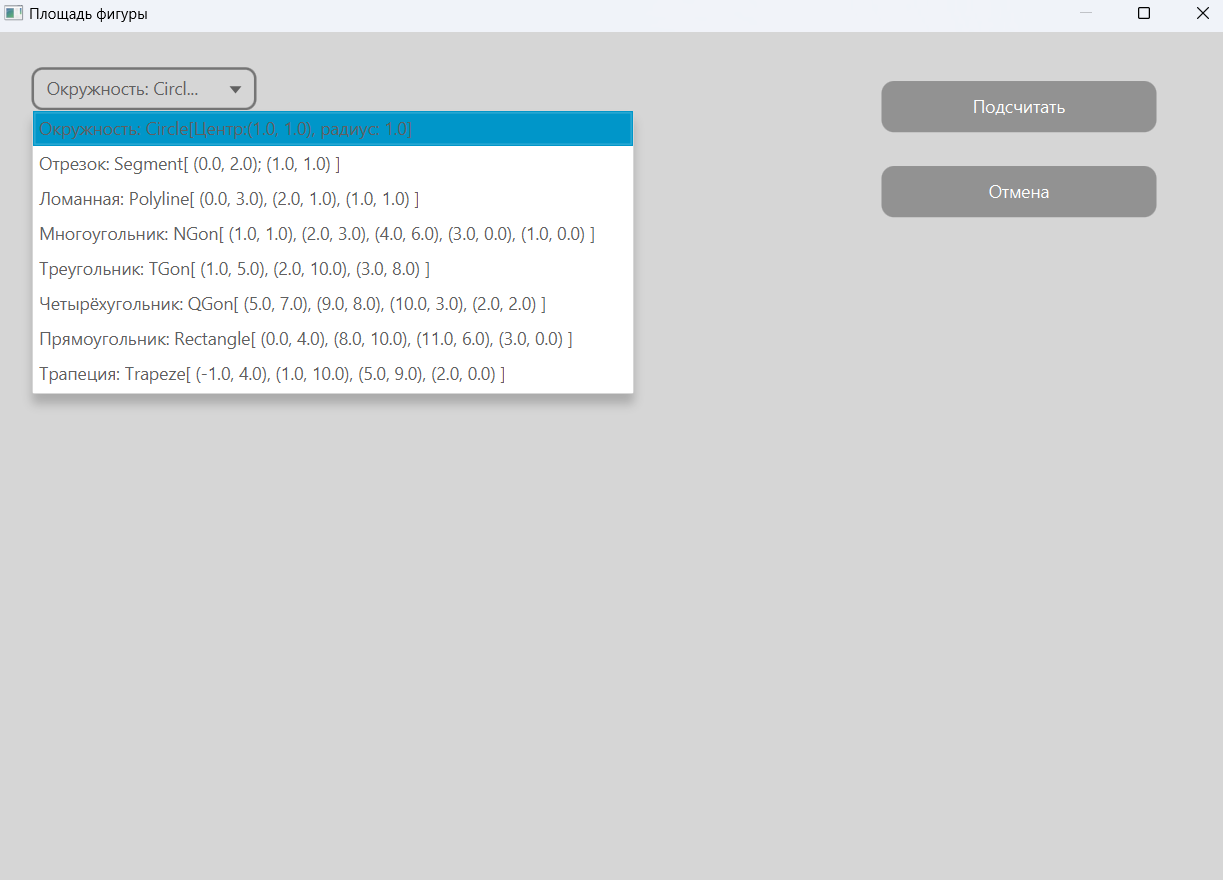


### **Пункт “Очистить”**

Результат

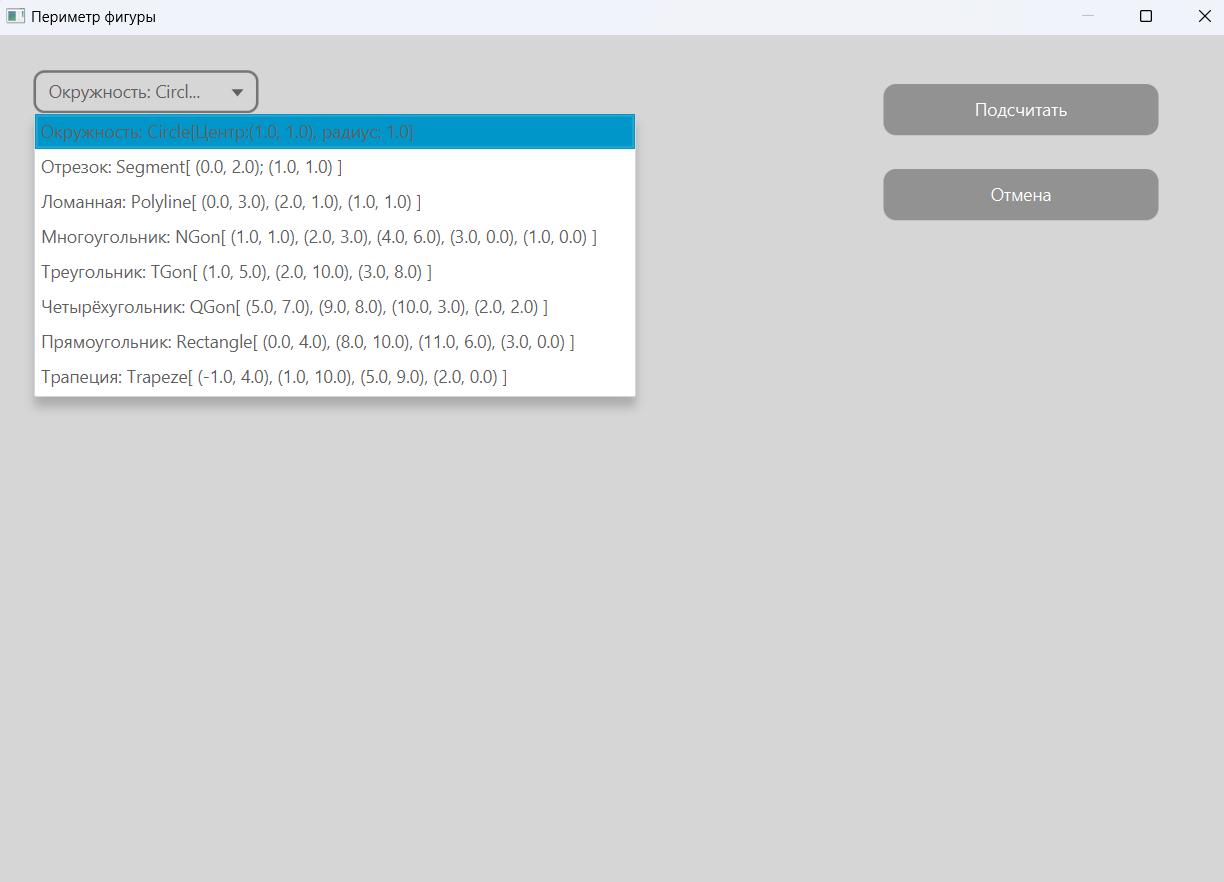


### **Пункт “Площадь фигуры”**

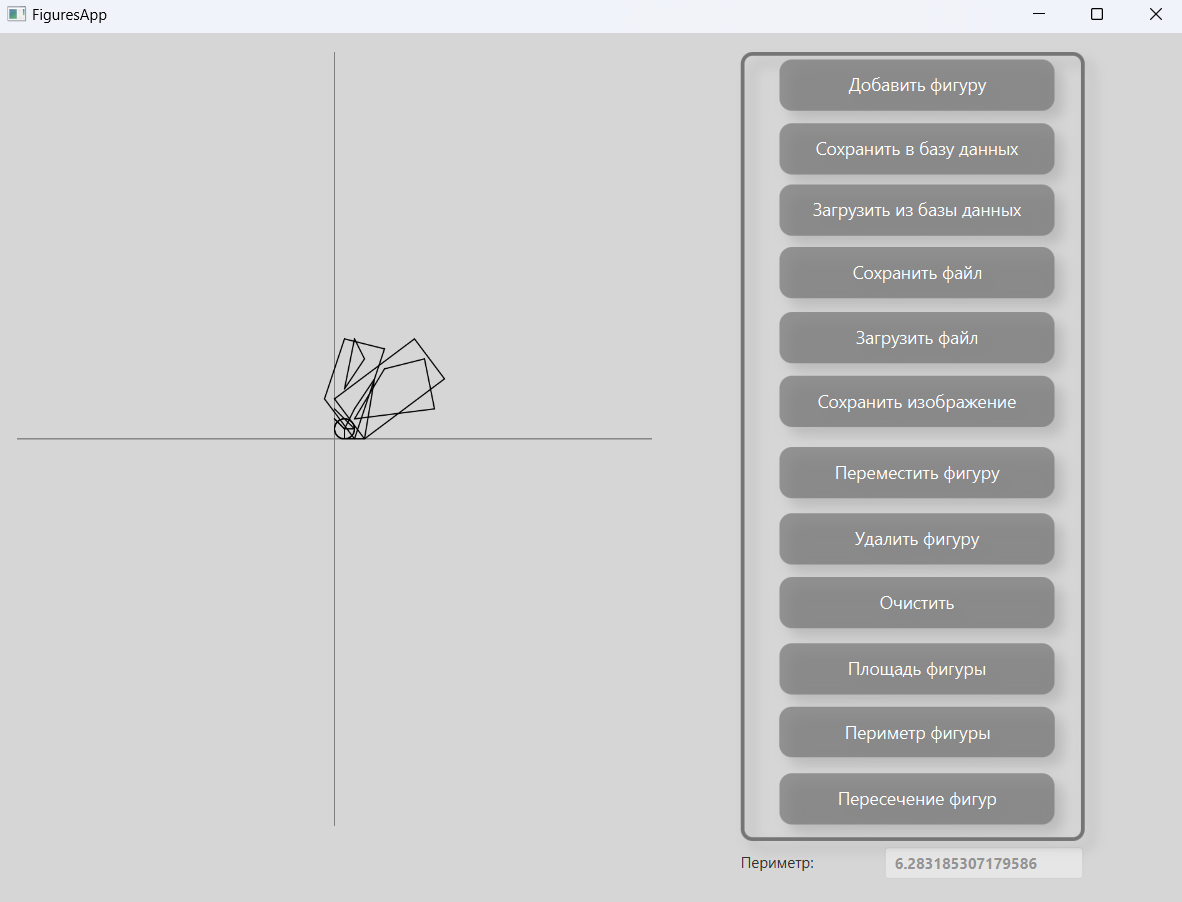
  
Результат



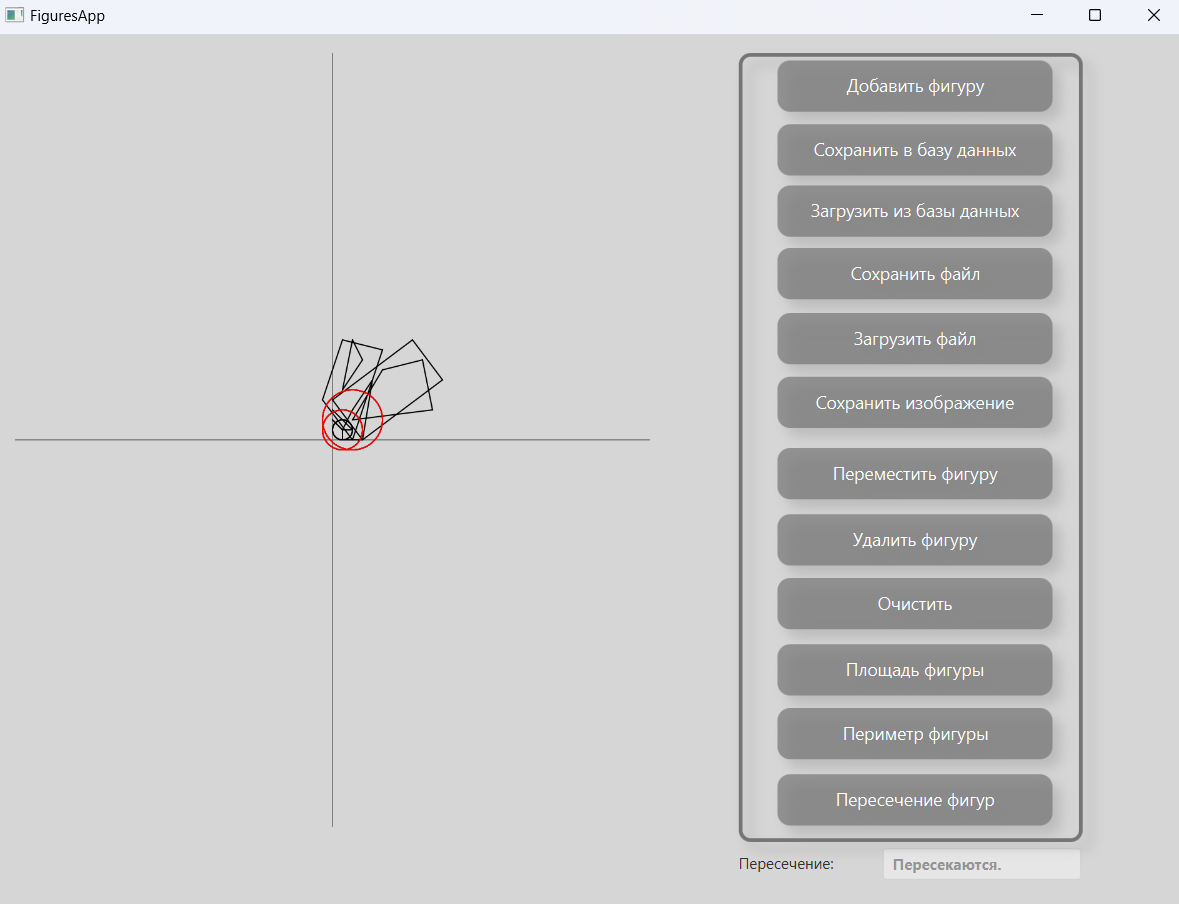
### **Пункт “Периметр фигуры”**



Результат



### **Пункт “Пересечение фигур”**



Пункты “Сохранить в базу данных” и “Загрузить из базы данных” были добавлены в четвертой части работы.

# Часть 3. Внедрение классов фигур с помощью .jar файла в JavaFX приложение. Тестирование классов

## **Цель**

* Протестировать все классы фигур с помощью фреймворка JUnit.
* Собрать первый проект в .jar файл.
* Внедрить первый проект во второй как зависимость.

## **Ход работы**

Загрузка и добавление зависимости

<dependency>  
 <groupId>org.junit.jupiter</groupId>  
 <artifactId>junit-jupiter</artifactId>  
 <version>5.9.2</version>  
 <scope>test</scope>  
</dependency>

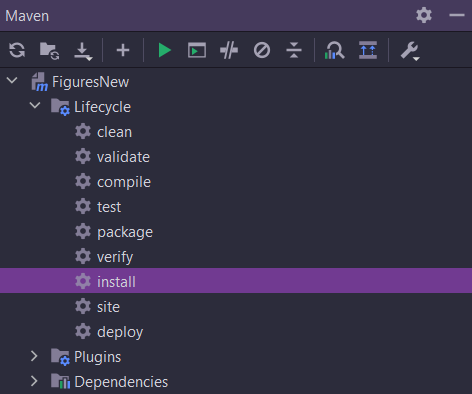
Создание классов-тестов и тестирование абсолютно всех методов классов фигур.

Использование различных ассертов, использование параметризованных тестов для осуществления некоторой оптимизации.

Пример:

@Test  
void testConstructorWithInvalidRadius() {  
 Point2D center = new Point2D()**;** double invalidRadius = 0**;** *assertThrows*(IllegalArgumentException.class**,** () -> new Circle(center**,** invalidRadius)**,** "Конструктор Circle должен выбрасывать исключение при недопустимом радиусе")**;**}

С помощью Maven архивирование первого проекта в java-архив .jar и его загрузка в локальный Maven-репозиторий .m2



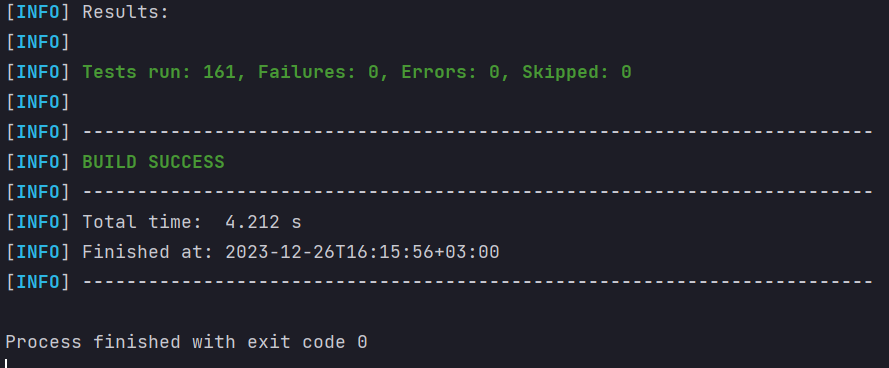
Во втором проекте подгружаем первый как зависимость в файле pom.xml

<dependency>  
 <groupId>org.figures</groupId>  
 <artifactId>FiguresNew</artifactId>  
 <version>1.0.0</version>  
</dependency>

Используем все классы фигур из первого проекта во втором.

## **Результат**

Все методы были успешно протестированы с помощью фреймворка Junit



Проект JavaFX успешно использует классы из первого проекта.

import org.figures.figures.\***;**import org.figures.figures.points.Point**;**import org.figures.figures.points.Point2D**;**import org.figures.figures.points.Point3D**;**import org.figures.interfaces.IShape**;**

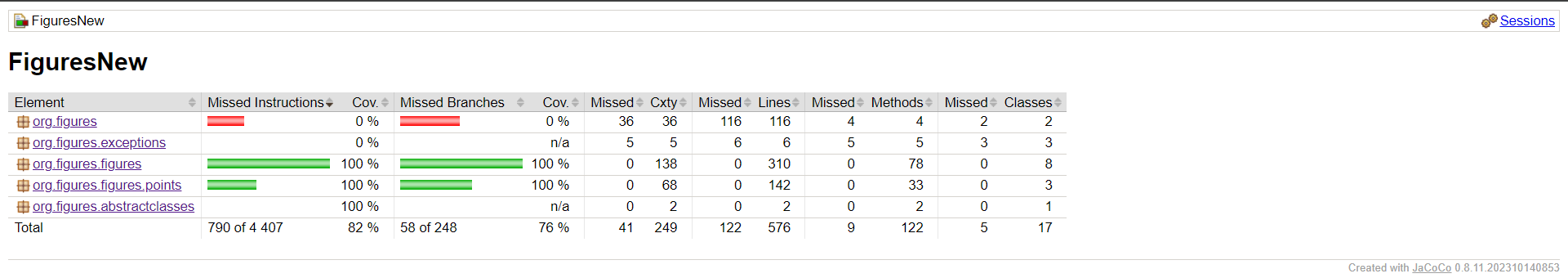
# Часть 4. Проверка покрытия кода тестами. Проверка на уязвимости и баги. Внедрение аннотаций. Внедрение БД

## **Описание этапов**

Внедрение библиотеки JaCoCo

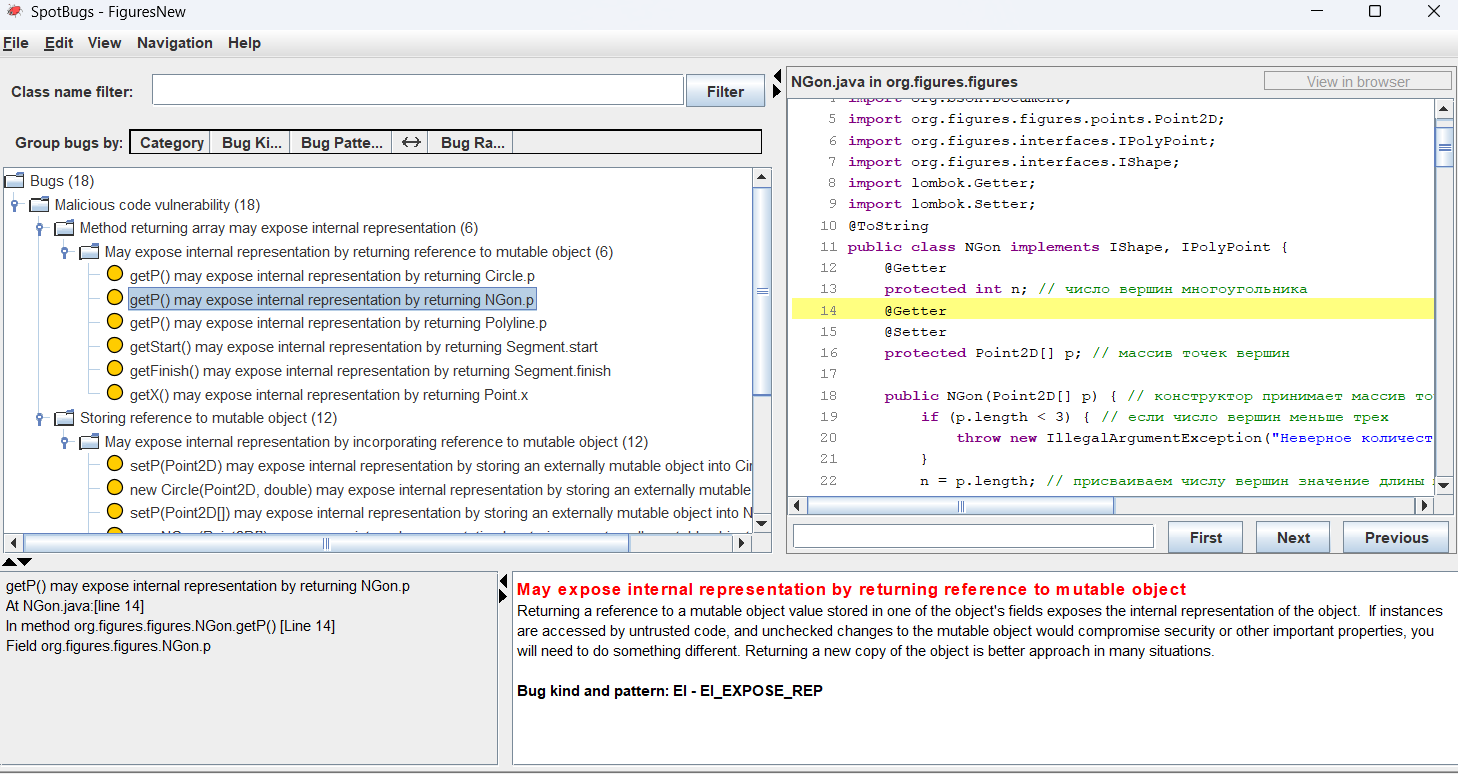
<plugin>  
 <groupId>org.jacoco</groupId>  
 <artifactId>jacoco-maven-plugin</artifactId>  
 <version>0.8.11</version>  
 <executions>  
 <execution>  
 <goals>  
 <goal>prepare-agent</goal>  
 </goals>  
 </execution>  
 <execution>  
 <id>report</id>  
 <phase>prepare-package</phase>  
 <goals>  
 <goal>report</goal>  
 </goals>  
 </execution>  
 </executions>  
</plugin>

Проверка покрытия тестами всех классов **фигур**. Результат – 100% покрытие тестами.



Внедрение SpotBugs. Проверка на уязвимости.

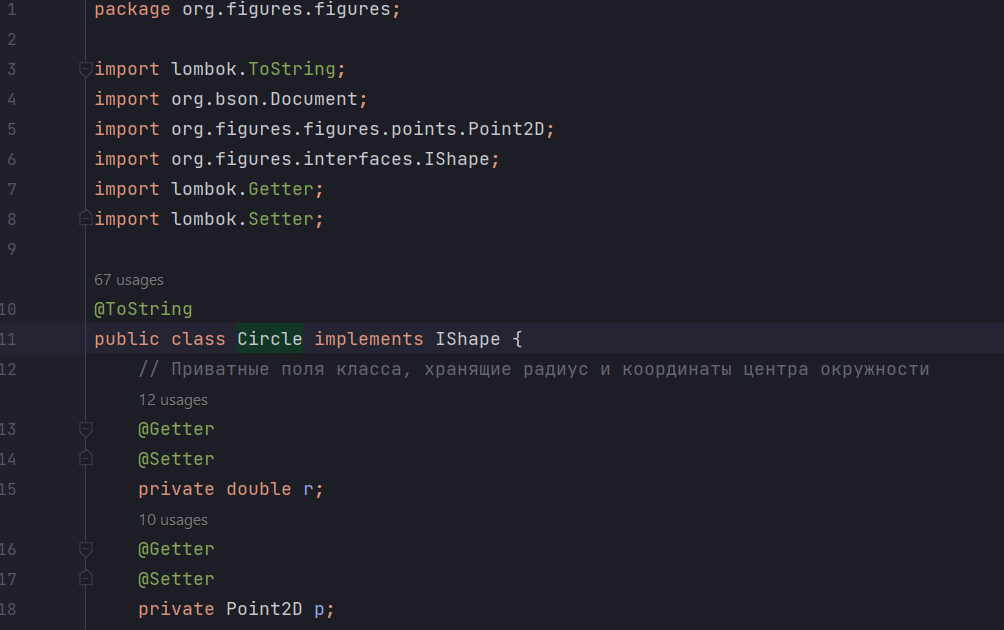
<plugin>  
 <groupId>com.github.spotbugs</groupId>  
 <artifactId>spotbugs-maven-plugin</artifactId>  
 <version>4.7.3.6</version>  
 <dependencies>  
 <dependency>  
 <groupId>com.github.spotbugs</groupId>  
 <artifactId>spotbugs</artifactId>  
 <version>4.7.3</version>  
 </dependency>  
 </dependencies>  
</plugin>



Незначительные уязвимости, связанные с get и set методами, а так же присваиванием переменных в конструкторах. В нашем случае данные уязвимости не являются потенциальной угрозой.

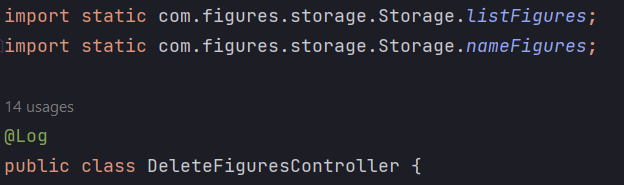
Внедрение Lombok и аннотаций.

Пример:

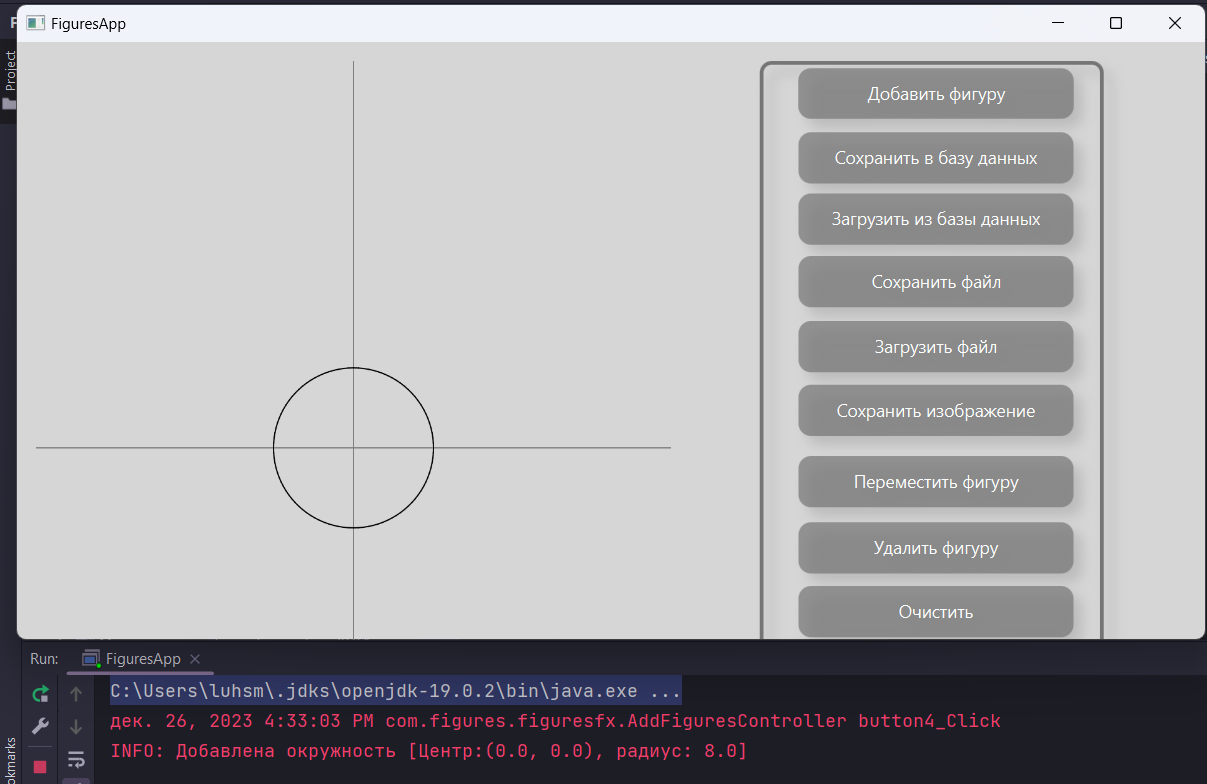


Были внедрены аннотации @ToString @Getter @Setter, которые автоматически переопределяют соответствующие методы.

В JavaFX проекте была внедрена аннотация @Log, обеспечивающее логирование некоторых процессов



Пример:



Код:

*log*.info("Добавлена окружность " + circle)**;**

Были добавлены методы перевода объектов в .json формат

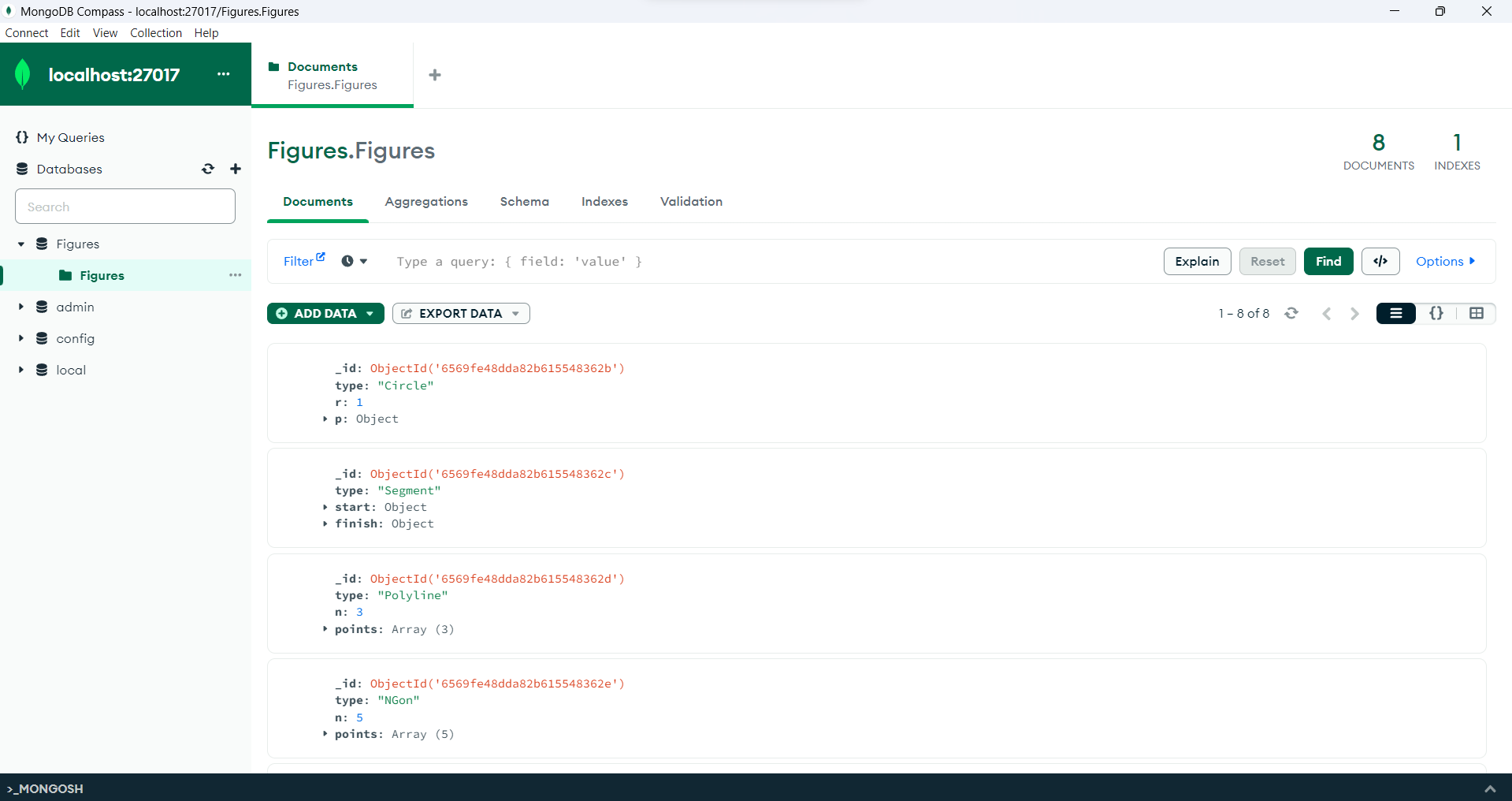
Пример:

public Document toDocument() {  
 Document document = new Document()**;** document.append("type"**,** "Circle")**;** document.append("r"**,** r)**;** document.append("p"**,** p.toDocument())**;** return document**;**}

Добавлены методы сохранения и выгрузки объектов в\из базы данных.

Код:

//Загрузка в базу данных  
public void button\_loadINBD\_Click(ActionEvent actionEvent) {  
  
 try {  
 CodecRegistry codecRegistry = CodecRegistries.*fromRegistries*(  
 MongoClientSettings.*getDefaultCodecRegistry*()**,** CodecRegistries.*fromCodecs*(new DoubleArrayCodec())  
 )**;** PojoCodecProvider pojoCodecProvider = PojoCodecProvider.*builder*().automatic(true).build()**;** CodecRegistry pojoCodecRegistry = CodecRegistries.*fromProviders*(pojoCodecProvider)**;** CodecRegistry combinedCodecRegistry = CodecRegistries.*fromRegistries*(codecRegistry**,** pojoCodecRegistry)**;** String connectionString = "mongodb://localhost:27017"**;** ConnectionString connString = new ConnectionString(connectionString)**;** MongoClientSettings settings = MongoClientSettings.*builder*()  
 .applyConnectionString(connString)  
 .codecRegistry(combinedCodecRegistry)  
 .build()**;** try (MongoClient mongoClient = MongoClients.*create*(settings)) {  
 MongoDatabase database = mongoClient.getDatabase("Figures")**;** // Очистка коллекции перед добавлением новых фигур  
 MongoCollection<Document> collection = database.getCollection("Figures")**;** collection.deleteMany(new Document())**;** // Цикл по всем фигурам  
 for (int i = 0**;** i < Storage.*listFigures*.size()**;** i++) {  
 IShape figure = Storage.*listFigures*.get(i)**;** Document document = createFigureDocument(figure)**;** collection.insertOne(document)**;** }  
 }  
 } catch (Exception e) {  
 e.printStackTrace()**;** }  
}  
  
// Загрузка из БД  
public void Button\_loadFromBD\_Click(ActionEvent actionEvent) {  
 try {  
 Storage.*listFigures*.clear()**;** Storage.*nameFigures*.clear()**;** Storage.*figuresCount* = 0**;** CodecRegistry codecRegistry = CodecRegistries.*fromRegistries*(  
 MongoClientSettings.*getDefaultCodecRegistry*()**,** CodecRegistries.*fromCodecs*(new DoubleArrayCodec())  
 )**;** PojoCodecProvider pojoCodecProvider = PojoCodecProvider.*builder*().automatic(true).build()**;** CodecRegistry pojoCodecRegistry = CodecRegistries.*fromProviders*(pojoCodecProvider)**;** CodecRegistry combinedCodecRegistry = CodecRegistries.*fromRegistries*(codecRegistry**,** pojoCodecRegistry)**;** String connectionString = "mongodb://localhost:27017"**;** ConnectionString connString = new ConnectionString(connectionString)**;** MongoClientSettings settings = MongoClientSettings.*builder*()  
 .applyConnectionString(connString)  
 .codecRegistry(combinedCodecRegistry)  
 .build()**;** try (MongoClient mongoClient = MongoClients.*create*(settings)) {  
 MongoDatabase database = mongoClient.getDatabase("Figures")**;** // Получаем коллекцию  
 MongoCollection<Document> collection = database.getCollection("Figures")**;** // Находим все документы в коллекции  
 FindIterable<Document> documents = collection.find()**;** // Очищаем список фигур перед загрузкой новых  
 Storage.*listFigures*.clear()**;** // Проходим по документам и создаем объекты фигур  
 for (Document document : documents) {  
 IShape figure = createFigureFromDocument(document)**;** // Добавляем фигуру в список фигур  
 Storage.*listFigures*.add(figure)**;** Storage.*figuresCount*++**;** }  
 }  
 GraphicsContext gc = canvas.getGraphicsContext2D()**;** gc.clearRect(0**,** 0**,** canvas.getWidth()**,** canvas.getHeight())**;** application.drawAxes(canvas.getWidth()**,** canvas.getHeight())**;** for(int i = 0**;** i < Storage.*figuresCount***;** i++){  
 draw(Storage.*getListFigures*().get(i)**,** Color.*BLACK*)**;** }  
 } catch (Exception e) {  
 e.printStackTrace()**;** }  
}



# Листинг кода

## **Point**

**package** org.figures.figures.points;

**import** **lombok.Getter**;

**import** **lombok.ToString**;

**import** **org.bson.Document**;

**@ToString**

**public** **class** **Point** {

**@Getter**

**protected** **int** dim; // Размерность пространства

**@Getter**

**protected** **double**[] x; // Массив координат точки

// Конструктор класса с одним параметром - размерностью пространства

**public** **Point**(**int** dim) {

// Инициализация поля dim переданным значением

**this**.dim = dim;

// Создание массива x размерности dim

x = **new** **double**[dim];

}

// Конструктор класса с двумя параметрами - размерностью пространства и массивом координат

**public** **Point**(**int** dim, **double**[] x) {

// Проверка соответствия размерности массива x заданной размерности пространства

**if** (dim != x.length) {

// Выброс исключения с сообщением о неправильной размерности массива

**throw** **new** **IllegalArgumentException**("Размерность массива не совпадает с заданной размерностью пространства.");

}

// Инициализация полей dim и x переданными значениями

**this**.dim = dim;

**this**.x = x;

}

// Метод для получения значения i-й координаты точки

**public** **double** **getX**(**int** i) {

// Проверка правильности переданного индекса

**if** (i < **0** || i >= dim) {

// Выброс исключения с сообщением о неправильном индексе

**throw** **new** **IllegalArgumentException**("Неверный индекс координаты.");

}

**return** x[i];

}

// Метод для установки нового массива координат точки

**public** **void** **setX**(**double**[] x) {

// Проверка соответствия размерности переданного массива x заданной размерности пространства

**if** (dim != x.length) {

// Выброс исключения с сообщением о неправильной размерности массива

**throw** **new** **IllegalArgumentException**("Размерность массива не совпадает с заданной размерностью пространства.");

}

**this**.x = x;

}

// Метод для установки нового значения i-й координаты точки

**public** **void** **setX**(**double** x, **int** i) {

// Проверка правильности переданного индекса

**if** (i < **0** || i >= dim) {

// Выброс исключения с сообщением о неправильном индексе

**throw** **new** **IllegalArgumentException**("Неверный индекс координаты.");

}

**this**.x[i] = x;

}

// метод для вычисления модуля точки(расстояние от начала координат)

**public** **double** **abs**() {

// инициализируем переменную суммой квадратов координат точки

**double** sum = **0**;

**for** (**int** i = **0**; i < dim; i++) {

// квадрат координаты i-ой оси добавляется к сумме

sum += x[i] \* x[i];

}

// вычисляем корень суммы квадратов координат и возвращаем его

**return** Math.sqrt(sum);

}

// сумма точек

**public** **static** Point **add**(Point a, Point b) {

// проверяем, что размерности точек совпадают

**if** (a.dim != b.dim) {

**throw** **new** **IllegalArgumentException**("Размерности точек не совпадают.");

}

// инициализируем массив координат результата

**double**[] res = **new** **double**[a.dim];

**for** (**int** i = **0**; i < a.dim; i++) {

// складываем координаты a и b на каждой оси и сохраняем результат в массив res

res[i] = a.x[i] + b.x[i];

}

// создаем новую точку с координатами из массива res и возвращаем ее

**return** **new** **Point**(a.dim, res);

}

**public** Point **add**(Point b) {

// проверяем, что размерности точек совпадают

**if** (dim != b.dim) {

**throw** **new** **IllegalArgumentException**("Размерности точек не совпадают.");

}

// инициализируем массив координат результата

**double**[] res = **new** **double**[dim];

**for** (**int** i = **0**; i < dim; i++) {

// складываем координаты текущей точки и точки b на каждой оси и сохраняем результат в массив res

res[i] = x[i] + b.x[i];

}

// создаем новую точку с координатами из массива res и возвращаем ее

**return** **new** **Point**(dim, res);

}

// разность точек

**public** **static** Point **sub**(Point a, Point b) {

// проверяем, что размерности точек совпадают

**if** (a.dim != b.dim) {

**throw** **new** **IllegalArgumentException**("Размерности точек не совпадают.");

}

// инициализируем массив координат результата

**double**[] res = **new** **double**[a.dim];

**for** (**int** i = **0**; i < a.dim; i++) {

// вычитаем координаты b из координат a на каждой оси и сохраняем результат в массив res

res[i] = a.x[i] - b.x[i];

}

// создаем новую точку с координатами из массива res и возвращаем ее

**return** **new** **Point**(a.dim, res);

}

**public** Point **sub**(Point b) {

// проверяем, что размерности точек совпадают

**if** (dim != b.dim) {

**throw** **new** **IllegalArgumentException**("Размерности точек не совпадают.");

}

// создаем массив для хранения разности координат

**double**[] res = **new** **double**[dim];

// вычисляем разность координат

**for** (**int** i = **0**; i < dim; i++) {

res[i] = x[i] - b.x[i];

}

// возвращаем новую точку с полученными координатами

**return** **new** **Point**(dim, res);

}

// умножение точки на число

**public** **static** Point **mult**(Point a, **double** r) {

// создаем массив для хранения результата умножения

**double**[] res = **new** **double**[a.dim];

// умножаем каждую координату на заданный коэффициент

**for** (**int** i = **0**; i < a.dim; i++) {

res[i] = a.x[i] \* r;

}

// возвращаем новую точку с полученными координатами

**return** **new** **Point**(a.dim, res);

}

**public** Point **mult**(**double** r) {

// создаем массив для хранения результата умножения

**double**[] res = **new** **double**[dim];

// умножаем каждую координату на заданный коэффициент

**for** (**int** i = **0**; i < dim; i++) {

res[i] = x[i] \* r;

}

// возвращаем новую точку с полученными координатами

**return** **new** **Point**(dim, res);

}

**public** **static** **double** **mult**(Point a, Point b) {

// проверяем, что размерности точек совпадают

**if** (a.dim != b.dim) {

**throw** **new** **IllegalArgumentException**("Размерности точек не совпадают.");

}

// вычисляем скалярное произведение

**double** res = **0**;

**for** (**int** i = **0**; i < a.dim; i++) {

res += a.x[i] \* b.x[i];

}

// возвращаем результат

**return** res;

}

**public** **double** **mult**(Point b) {

**if** (dim != b.dim) {

**throw** **new** **IllegalArgumentException**("Размерности точек не совпадают.");

}

**double** res = **0**;

**for** (**int** i = **0**; i < dim; i++) {

res += x[i] \* b.x[i];

}

**return** res;

}

// симметричное отражение относительно i-ой оси

**public** **static** Point **symAxis**(Point a, **int** i) {

// Создаем новый массив для точки, чтобы не изменять исходный массив.

**double**[] res = **new** **double**[a.dim];

**for** (**int** j = **0**; j < a.dim; j++) {

res[j] = a.x[j];

}

// Проверяем, что номер оси не выходит за границы размерности точки.

**if** (i < **0** || i >= a.dim) {

**throw** **new** **IllegalArgumentException**("Неверный номер оси.");

}

// Симметрия точки относительно оси путем инвертирования всех координат точки, кроме координаты, соответствующей оси симметрии.

**for** (**int** k = **0**; k < a.dim; k++) {

**if** (k != i) {

res[k] = -res[k];

}

}

// Возвращаем новую точку, полученную после симметрии.

**return** **new** **Point**(a.dim, res);

}

**public** Point **symAxis**(**int** i) {

// Создаем новый массив для точки, чтобы не изменять исходный массив.

**double**[] res = **new** **double**[dim];

**for** (**int** j = **0**; j < dim; j++) {

res[j] = x[j];

}

// Проверяем, что номер оси не выходит за границы размерности точки.

**if** (i < **0** || i >= dim) {

**throw** **new** **IllegalArgumentException**("Неверный номер оси.");

}

// Симметрия точки относительно оси путем инвертирования всех координат точки, кроме координаты, соответствующей оси симметрии.

**for** (**int** k = **0**; k < dim; k++) {

**if** (k != i) {

res[k] = -res[k];

}

}

// Возвращаем новую точку, полученную после симметрии.

**return** **new** **Point**(dim, res);

}

**public** Document **toDocument**() {

Document document = **new** Document();

document.append("type", "Point");

document.append("dim", dim);

document.append("x", x);

**return** document;

}

}

## **Point2D**

**package** org.figures.figures.points;

**import** **org.bson.Document**;

**public** **class** **Point2D** **extends** Point {

// Конструктор без параметров, создающий двумерную точку

**public** **Point2D**() {

**super**(**2**);

}

// Конструктор, принимающий массив координат, создающий двумерную точку

**public** **Point2D**(**double**[] x) {

**super**(**2**, x);

}

// Метод, возвращающий новую точку, полученную из данной поворотом на угол phi

**public** **static** Point2D **rot**(Point2D a, **double** phi) {

**double**[] res = **new** **double**[**2**];

res[**0**] = a.x[**0**] \* Math.cos(phi) - a.x[**1**] \* Math.sin(phi);

res[**1**] = a.x[**0**] \* Math.sin(phi) + a.x[**1**] \* Math.cos(phi);

**return** **new** **Point2D**(res);

}

// Метод, поворачивающий данную точку на угол phi и возвращающий новую точку

**public** Point2D **rot**(**double** phi) {

**double**[] res = **new** **double**[**2**];

res[**0**] = x[**0**] \* Math.cos(phi) - x[**1**] \* Math.sin(phi);

res[**1**] = x[**0**] \* Math.sin(phi) + x[**1**] \* Math.cos(phi);

**return** **new** **Point2D**(res);

}

// Метод, возвращающий новую точку, полученную из данной симметрией относительно оси i

**@Override**

**public** Point **symAxis**(**int** i) {

**double**[] res = **new** **double**[**2**];

**for** (**int** j = **0**; j < dim; j++) {

res[j] = x[j];

}

**if** (i < **0** || i >= **2**) {

**throw** **new** **IllegalArgumentException**("Неверный номер оси.");

}

**if** (i == **0**)

res[**1**] = -res[**1**];

**else**

res[**0**] = -res[**0**];

**return** **new** **Point2D**(res);

}

**public** Document **toDocument**() {

Document document = **new** Document();

document.append("type", "Point2D");

document.append("dim", dim);

document.append("x", x);

**return** document;

}

}

## **Point3D**

**package** org.figures.figures.points;

**import** **org.bson.Document**;

**public** **class** **Point3D** **extends** Point {

**public** **Point3D**() {

**super**(**3**);

}

**public** **Point3D**(**double**[] x) {

**super**(**3**, x);

}

// Статический метод, вычисляющий векторное произведение двух векторов типа Classes.Point3D

**public** **static** Point3D **cross\_prod**(Point3D p1, Point3D p2) {

**double**[] res = **new** **double**[**3**];

res[**0**] = p1.x[**1**] \* p2.x[**2**] - p1.x[**2**] \* p2.x[**1**];

res[**1**] = p1.x[**2**] \* p2.x[**0**] - p1.x[**0**] \* p2.x[**2**];

res[**2**] = p1.x[**0**] \* p2.x[**1**] - p1.x[**1**] \* p2.x[**0**];

**return** **new** **Point3D**(res);

}

// Метод, вычисляющий векторное произведение текущего объекта типа Classes.Point3D и переданного в метод объекта типа Classes.Point3D

**public** Point3D **cross\_prod**(Point3D p) {

**double**[] res = **new** **double**[**3**];

res[**0**] = x[**1**] \* p.x[**2**] - x[**2**] \* p.x[**1**];

res[**1**] = x[**2**] \* p.x[**0**] - x[**0**] \* p.x[**2**];

res[**2**] = x[**0**] \* p.x[**1**] - x[**1**] \* p.x[**0**];

**return** **new** **Point3D**(res);

}

// Статический метод, вычисляющий смешанное произведение трех векторов типа Classes.Point3D

**public** **static** **double** **mix\_prod**(Point3D p1, Point3D p2, Point3D p3) {

**return** p1.x[**0**] \* (p2.x[**1**] \* p3.x[**2**] - p3.x[**1**] \* p2.x[**2**])

+ p1.x[**1**] \* (p2.x[**2**] \* p3.x[**0**] - p3.x[**2**] \* p2.x[**0**])

+ p1.x[**2**] \* (p2.x[**0**] \* p3.x[**1**] - p3.x[**0**] \* p2.x[**1**]);

}

// Метод, вычисляющий смешанное произведение текущего объекта типа Classes.Point3D и двух переданных в метод объектов типа Classes.Point3D

**public** **double** **mix\_prod**(Point3D p1, Point3D p2) {

**return** x[**0**] \* (p1.x[**1**] \* p2.x[**2**] - p2.x[**1**] \* p1.x[**2**])

+ x[**1**] \* (p1.x[**2**] \* p2.x[**0**] - p2.x[**2**] \* p1.x[**0**])

+ x[**2**] \* (p1.x[**0**] \* p2.x[**1**] - p2.x[**0**] \* p1.x[**1**]);

}

**public** Document **toDocument**() {

Document document = **new** Document();

document.append("type", "Point3D");

document.append("dim", dim);

document.append("x", x);

**return** document;

}

}

## **Circle**

**package** org.figures.figures;

**import** **lombok.ToString**;

**import** **org.bson.Document**;

**import** **org.figures.figures.points.Point2D**;

**import** **org.figures.interfaces.IShape**;

**import** **lombok.Getter**;

**import** **lombok.Setter**;

**@ToString**

**public** **class** **Circle** **implements** IShape {

// Приватные поля класса, хранящие радиус и координаты центра окружности

**@Getter**

**@Setter**

**private** **double** r;

**@Getter**

**@Setter**

**private** Point2D p;

// Конструктор класса Classes.Circle, принимающий координаты центра окружности и ее радиус

**public** **Circle**(Point2D p, **double** r) {

// Проверка на правильность введенных данных

**if** (r <= **0**) {

**throw** **new** **IllegalArgumentException**("Неверный радиус.");

}

// Присвоение значений полям

**this**.p = p;

**this**.r = r;

}

// Метод для вычисления площади окружности

**public** **double** **square**() {

**return** Math.PI \* r \* r;

}

// Метод для вычисления длины окружности

**public** **double** **length**() {

**return** **2** \* Math.PI \* r;

}

// Метод для перемещения окружности на заданный вектор

**public** IShape **shift**(Point2D a) {

**return** **new** **Circle**(**new** Point2D(**new** **double**[]{p.getX(**0**) + a.getX(**0**), p.getX(**1**) + a.getX(**1**)}), r);

}

// Метод для поворота окружности на заданный угол

**public** IShape **rot**(**double** phi) {

**return** **new** **Circle**(p.rot(phi), r);

}

// Метод для получения симметричной окружности относительно заданной оси

**public** IShape **symAxis**(**int** i) {

**return** **new** **Circle**((Point2D) p.symAxis(i), r);

}

// Метод для проверки пересечения данной окружности с другой фигурой

**public** **boolean** **cross**(IShape i) {

// Если входной параметр является экземпляром класса Classes.Circle, то выполняется следующее

**if** (i **instanceof** Circle) {

// Приведение входного параметра к типу Classes.Circle

Circle c = (Circle) i;

// Вычисление расстояния между центрами окружностей

**double** res = Math.sqrt(Math.pow(p.getX(**0**) - c.p.getX(**0**), **2**) + Math.pow(p.getX(**1**) - c.p.getX(**1**), **2**));

// Проверка на пересечение окружностей

**return** res < (r + c.r) && res > Math.abs(r - c.r);

} **else** {

**throw** **new** **IllegalArgumentException**("Аргумент должен быть экземпляром класса Classes.Circle");

}

}

**public** Document **toDocument**() {

Document document = **new** Document();

document.append("type", "Circle");

document.append("r", r);

document.append("p", p.toDocument());

**return** document;

}

}

## **NGon**

**package** org.figures.figures;

**import** **lombok.ToString**;

**import** **org.bson.Document**;

**import** **org.figures.figures.points.Point2D**;

**import** **org.figures.interfaces.IPolyPoint**;

**import** **org.figures.interfaces.IShape**;

**import** **lombok.Getter**;

**import** **lombok.Setter**;

**@ToString**

**public** **class** **NGon** **implements** IShape, IPolyPoint {

**@Getter**

**protected** **int** n; // число вершин многоугольника

**@Getter**

**@Setter**

**protected** Point2D[] p; // массив точек вершин

**public** **NGon**(Point2D[] p) { // конструктор принимает массив точек вершин

**if** (p.length < **3**) { // если число вершин меньше трех

**throw** **new** **IllegalArgumentException**("Неверное количество углов."); // выбрасываем исключение с сообщением об ошибке

}

n = p.length; // присваиваем числу вершин значение длины массива точек вершин

**this**.p = p; // сохраняем массив точек вершин

}

**public** Point2D **getP**(**int** i) { // метод возвращает i-ую точку вершины многоугольника

**if** (i < **0** || i >= n) { // если i не попадает в диапазон от 0 до n-1

**throw** **new** **IllegalArgumentException**("Неверный индекс координаты."); // выбрасываем исключение с сообщением об ошибке

}

**return** p[i]; // возвращаем i-ую точку вершины

}

**public** **void** **setP**(Point2D p, **int** i) { // метод устанавливает новые координаты i-ой вершины многоугольника

**if** (i < **0** || i >= n) { // если i не попадает в диапазон от 0 до n-1

**throw** **new** **IllegalArgumentException**("Неверный индекс координаты."); // выбрасываем исключение с сообщением об ошибке

}

**this**.p[i] = p; // сохраняем новую координату i-ой вершины

}

// Определяется метод для вычисления площади многоугольника (ПО ФОРМУЛЕ ГАУССА ЧЕРЕЗ ОПРЕДЕЛИТЕЛЬ МАТРИЦЫ)

**public** **double** **square**() {

**double** res = **0**; // Инициализируется переменная для хранения результата

**double** ox = p[**0**].getX(**0**); // Получаем значение координаты x первой вершины

**double** oy = p[**0**].getX(**1**); // Получаем значение координаты y первой вершины

Point2D[] t = **new** Point2D[n + **1**]; // Создается массив для хранения вершин и добавляется место для первой вершины в конце

// Копируем вершины из исходного массива в массив t

**for** (**int** i = **0**; i < n; i++) {

t[i] = p[i];

}

t[n] = p[**0**];

// Проходим по всем вершинам и вычисляем площадь

**for** (**int** i = **1**; i < n + **1**; i++) {

**double** x = t[i].getX(**0**);

**double** y = t[i].getX(**1**);

res += (x \* oy - y \* ox); // Добавляем текущее значение в сумму

ox = x; // Обновляем значение координаты x

oy = y; // Обновляем значение координаты y

}

**return** Math.abs(res / **2**); // Возвращаем значение площади, поделенной на 2

}

// Определяется метод для вычисления периметра многоугольника

**public** **double** **length**() {

**double** res = **0**; // Инициализируется переменная для хранения результата

// Проходим по всем ребрам и вычисляем длину

**for** (**int** i = **0**; i < n - **1**; i++) {

**double** dx = p[i + **1**].getX(**0**) - p[i].getX(**0**); // Вычисляем разницу координат x между соседними вершинами

**double** dy = p[i + **1**].getX(**1**) - p[i].getX(**1**); // Вычисляем разницу координат y между соседними вершинами

res += Math.sqrt(dx \* dx + dy \* dy); // Добавляем длину ребра в сумму

}

// Вычисляем длину последнего ребра, соединяющего последнюю и первую вершины

**double** l1 = p[n - **1**].getX(**0**) - p[**0**].getX(**0**);

**double** l2 = p[n - **1**].getX(**1**) - p[**0**].getX(**1**);

res += Math.sqrt(l1 \* l1 + l2 \* l2);

**return** res; // Возвращаем значение периметра

}

// Метод для перемещения фигуры на заданный вектор

**public** IShape **shift**(Point2D a) {

Point2D[] res = **new** Point2D[n];

// Цикл для перебора всех вершин фигуры

**for** (**int** i = **0**; i < n; i++) {

// Вычисляем новое положение вершины после сдвига на вектор a

res[i] = **new** Point2D(**new** **double**[]{p[i].getX(**0**) + a.getX(**0**), p[i].getX(**1**) + a.getX(**1**)});

}

// Возвращаем новую фигуру

**return** **new** **NGon**(res);

}

// Метод для поворота фигуры на заданный угол

**public** IShape **rot**(**double** phi) {

Point2D[] res = **new** Point2D[n];

// Цикл для перебора всех вершин фигуры

**for** (**int** i = **0**; i < n; i++) {

// Вычисляем новое положение вершины после поворота на угол phi

res[i] = p[i].rot(phi);

}

// Возвращаем новую фигуру

**return** **new** **NGon**(res);

}

// Метод для отражения фигуры относительно заданной оси симметрии

**public** IShape **symAxis**(**int** i) {

Point2D[] res = **new** Point2D[n];

// Цикл для перебора всех вершин фигуры

**for** (**int** j = **0**; j < n; j++) {

// Вычисляем новое положение вершины после отражения относительно оси симметрии i

res[j] = (Point2D) p[j].symAxis(i);

}

// Возвращаем новую фигуру

**return** **new** **NGon**(res);

}

**public** **boolean** **cross**(IShape i) {

// Проверяем, что аргумент i является экземпляром класса Classes.NGon или его наследника

**if** (!(i **instanceof** NGon)) {

// Если i не является экземпляром класса Classes.NGon или его наследника, выбрасываем исключение IllegalArgumentException

**throw** **new** **IllegalArgumentException**("Аргумент должен быть экземпляром класса Classes.NGon или наследуемого от него класса");

}

// Приводим аргумент i к типу Classes.NGon и сохраняем в переменную other

NGon other = (NGon) i;

// Проходимся по всем вершинам другого многоугольника other

**for** (**int** j = **0**; j < other.getN(); j++) {

// Проходимся по всем вершинам текущего многоугольника

**for** (**int** k = **0**; k < n; k++) {

// Получаем координаты текущего отрезка (a, b) и отрезка другого многоугольника (c, d)

Point2D a = getP(k);

Point2D b = getP((k + **1**) % n);

Point2D c = other.getP(j);

Point2D d = other.getP((j + **1**) % other.getN());

// Проверяем пересекаются ли отрезки a-b и c-d

**if** (**new** Segment(a, b).cross(**new** Segment(c, d))) {

// Если отрезки пересекаются, возвращаем true

**return** **true**;

}

}

}

// Если ни один отрезок не пересекается, возвращаем false

**return** **false**;

}

**public** Document **toDocument**() {

Document document = **new** Document();

document.append("type", "NGon");

document.append("n", n);

// Создаем массив для точек вершин многоугольника

Document[] points = **new** Document[n];

**for** (**int** i = **0**; i < n; i++) {

points[i] = p[i].toDocument();

}

document.append("points", points);

**return** document;

}

}

## **Polyline**

**package** org.figures.figures;

**import** **lombok.ToString**;

**import** **org.bson.Document**;

**import** **org.figures.abstractclasses.OpenFigure**;

**import** **org.figures.figures.points.Point2D**;

**import** **org.figures.interfaces.IPolyPoint**;

**import** **org.figures.interfaces.IShape**;

**import** **lombok.Getter**;

**import** **lombok.Setter**;

**@ToString**

**public** **class** **Polyline** **extends** OpenFigure **implements** IPolyPoint {

**@Getter**

**private** **int** n;

**@Getter**

**@Setter**

**private** Point2D[] p;

// Конструктор класса, принимающий массив точек

**public** **Polyline**(Point2D[] p) {

// Задание количества точек

n = p.length;

// Задание массива точек

**this**.p = p;

}

// Метод для получения точки ломаной линии по индексу

**public** Point2D **getP**(**int** i) {

// Проверка на корректность индекса

**if** (i < **0** || i >= n) {

**throw** **new** **IllegalArgumentException**("Неверный индекс координаты.");

}

**return** p[i];

}

**public** **double** **square**(){

**return** **0**;

}

// Метод для задания точки ломаной линии по индексу

**public** **void** **setP**(Point2D p, **int** i) {

// Проверка на корректность индекса

**if** (i < **0** || i >= n) {

**throw** **new** **IllegalArgumentException**("Неверный индекс координаты.");

}

**this**.p[i] = p;

}

// Метод для вычисления длины ломаной линии

**@Override**

**public** **double** **length**() {

**double** res = **0**;

// Вычисление длины каждого отрезка и добавление к общей длине

**for** (**int** i = **0**; i < n - **1**; i++) {

**double** dx = p[i].getX(**0**) - p[i + **1**].getX(**0**);

**double** dy = p[i].getX(**1**) - p[i + **1**].getX(**1**);

res += Math.sqrt(dx \* dx + dy \* dy);

}

**return** res;

}

// Метод для перемещения ломаной линии на заданный вектор

**@Override**

**public** IShape **shift**(Point2D a) {

// Создание нового массива точек

Point2D[] res = **new** Point2D[n];

// Перемещение каждой точки линии на заданный вектор

**for** (**int** i = **0**; i < n; i++) {

res[i] = **new** Point2D(**new** **double**[]{p[i].getX(**0**) + a.getX(**0**), p[i].getX(**1**) + a.getX(**1**)});

}

// Создание новой ломаной линии с перемещенными точками

**return** **new** **Polyline**(res);

}

// Метод для поворота ломаной линии на заданный угол

**@Override**

**public** IShape **rot**(**double** phi) {

// Создание нового массива точек

Point2D[] res = **new** Point2D[n];

// Поворот каждой точки линии на заданный угол

**for** (**int** i = **0**; i < n; i++) {

res[i] = p[i].rot(phi);

}

// Создание новой ломаной линии с повернутыми точками

**return** **new** **Polyline**(res);

}

// Метод для выполнения симметрии ломаной линии относительно заданной оси

**@Override**

**public** IShape **symAxis**(**int** i) {

// Создание нового массива точек

Point2D[] res = **new** Point2D[n];

// Выполнение симметрии каждой точки линии относительно заданной оси

**for** (**int** j = **0**; j < n; j++) {

res[j] = (Point2D) p[j].symAxis(i);

}

// Создание новой ломаной линии с симметричными точками

**return** **new** **Polyline**(res);

}

// Переопределение метода проверки пересечения для класса Classes.Polyline

**@Override**

**public** **boolean** **cross**(IShape i) {

**if** (!(i **instanceof** Polyline)) {

**throw** **new** **IllegalArgumentException**("Аргумент должен быть экземпляром класса Classes.Polyline");

}

// Привести аргумент к типу Classes.Polyline

Polyline other = (Polyline) i;

// Проверить пересечения всех отрезков текущей и другой полилиний

**for** (**int** j = **0**; j < n - **1**; j++) {

**for** (**int** k = **0**; k < other.n - **1**; k++) {

// Получить координаты отрезков в формате начало-конец

Point2D p1 = p[j];

Point2D p2 = p[j + **1**];

Point2D q1 = other.p[k];

Point2D q2 = other.p[k + **1**];

**if** (**new** Segment(p1, p2).cross(**new** Segment(q1, q2))) {

**return** **true**;

}

}

}

// Если пересечений не найдено, вернуть false

**return** **false**;

}

**public** Document **toDocument**() {

Document document = **new** Document();

document.append("type", "Polyline");

document.append("n", n);

Document[] points = **new** Document[n];

**for** (**int** i = **0**; i < n; i++) {

points[i] = p[i].toDocument();

}

document.append("points", points);

**return** document;

}

}

QGon  
**package** org.figures.figures;

**import** **org.bson.Document**;

**import** **org.figures.figures.points.Point2D**;

**import** **org.figures.interfaces.IShape**;

**public** **class** **QGon** **extends** NGon {

**public** **QGon**(Point2D[] p) {

**super**(p);

}

// Переопределение метода вычисления площади четырехугольника

**@Override**

**public** **double** **square**() {

// Вычисление длин сторон многоугольника

**double** a = Math.sqrt(Math.pow(p[**1**].getX(**0**) - p[**0**].getX(**0**), **2**) + Math.pow(p[**1**].getX(**1**) - p[**0**].getX(**1**), **2**));

**double** b = Math.sqrt(Math.pow(p[**2**].getX(**0**) - p[**1**].getX(**0**), **2**) + Math.pow(p[**2**].getX(**1**) - p[**1**].getX(**1**), **2**));

**double** c = Math.sqrt(Math.pow(p[**0**].getX(**0**) - p[**2**].getX(**0**), **2**) + Math.pow(p[**0**].getX(**1**) - p[**2**].getX(**1**), **2**));

// Вычисление полупериметра многоугольника

**double** pr = (a + b + c) / **2**;

// Вычисление площади треугольника методом Герона

**double** pr1 = Math.sqrt(pr \* (pr - a) \* (pr - b) \* (pr - c));

a = Math.sqrt(Math.pow(p[**0**].getX(**0**) - p[**3**].getX(**0**), **2**) + Math.pow(p[**0**].getX(**1**) - p[**3**].getX(**1**), **2**));

b = Math.sqrt(Math.pow(p[**3**].getX(**0**) - p[**2**].getX(**0**), **2**) + Math.pow(p[**3**].getX(**1**) - p[**2**].getX(**1**), **2**));

// Вычисление полупериметра второго треугольника

pr = (a + b + c) / **2**;

// Вычисление площади второго треугольника методом Герона

**double** pr2 = Math.sqrt(pr \* (pr - a) \* (pr - b) \* (pr - c));

// Возвращение суммы площадей двух треугольников

**return** pr1 + pr2;

}

// Переопределение метода сдвига многоугольника на заданный вектор

**@Override**

**public** IShape **shift**(Point2D a) {

Point2D[] res = **new** Point2D[n];

**for** (**int** i = **0**; i < n; i++) {

res[i] = **new** Point2D(**new** **double**[]{p[i].getX(**0**) + a.getX(**0**), p[i].getX(**1**) + a.getX(**1**)});

}

**return** **new** **QGon**(res);

}

// Переопределение метода поворота многоугольника на заданный угол

**@Override**

**public** IShape **rot**(**double** phi) {

Point2D[] res = **new** Point2D[n];

**for** (**int** i = **0**; i < n; i++) {

res[i] = p[i].rot(phi);

}

**return** **new** **QGon**(res);

}

// Симметрично отражаем многоугольник относительно оси i

**@Override**

**public** IShape **symAxis**(**int** i) {

Point2D[] res = **new** Point2D[n];

**for** (**int** j = **0**; j < n; j++) {

res[j] = (Point2D) p[j].symAxis(i);

}

**return** **new** **QGon**(res);

}

**public** Document **toDocument**() {

Document document = **new** Document();

document.append("type", "QGon");

document.append("n", n);

Document[] points = **new** Document[n];

**for** (**int** i = **0**; i < n; i++) {

points[i] = p[i].toDocument();

}

document.append("points", points);

**return** document;

}

}

## **Rectangle**

**package** org.figures.figures;

**import** **org.bson.Document**;

**import** **org.figures.figures.points.Point2D**;

**import** **org.figures.interfaces.IShape**;

**public** **class** **Rectangle** **extends** QGon {

**public** **Rectangle**(Point2D[] p) {

**super**(p);

// if (!isRectangle()) {

// throw new IllegalArgumentException("Введите корректные данные для прямоугольника!");

// }

}

**@Override**

**public** **double** **square**() {

// Вычисление длины сторон прямоугольника по координатам вершин

**double** a = Math.sqrt(Math.pow(p[**1**].getX(**0**) - p[**0**].getX(**0**), **2**) + Math.pow(p[**1**].getX(**1**) - p[**0**].getX(**1**), **2**));

**double** b = Math.sqrt(Math.pow(p[**2**].getX(**0**) - p[**1**].getX(**0**), **2**) + Math.pow(p[**2**].getX(**1**) - p[**1**].getX(**1**), **2**));

// Вычисление площади прямоугольника

**return** a \* b;

}

/\*

public boolean isRectangle() {

// Проверяем, что противоположные стороны равны и параллельны

boolean parallel = false;

boolean equal = false;

for (int i = 0; i < 4; i++) {

int j = (i + 1) % 4;

int k = (i + 2) % 4;

int l = (i + 3) % 4;

if (p[i].getX(0) == p[j].getX(0) && p[k].getX(0) == p[l].getX(0)) {

if (p[i].getX(1) < p[j].getX(1)) {

if (p[k].getX(1) > p[l].getX(1) && p[k].getX(1) > p[i].getX(1) && p[l].getX(1) < p[j].getX(1)) {

parallel = true;

equal = (p[j].getX(1) - p[i].getX(1)) == (p[l].getX(1) - p[k].getX(1));

}

} else {

if (p[l].getX(1) > p[k].getX(1) && p[l].getX(1) > p[j].getX(1) && p[k].getX(1) < p[i].getX(1)) {

parallel = true;

equal = (p[i].getX(1) - p[j].getX(1)) == (p[k].getX(1) - p[l].getX(1));

}

}

}

}

return parallel && equal;

}

\*/

**@Override**

**public** IShape **shift**(Point2D a) {

// Создание массива вершин сдвинутого прямоугольника

Point2D[] res = **new** Point2D[n];

**for** (**int** i = **0**; i < n; i++) {

res[i] = **new** Point2D(**new** **double**[]{p[i].getX(**0**) + a.getX(**0**), p[i].getX(**1**) + a.getX(**1**)});

}

// Создание нового прямоугольника с измененными координатами вершин

**return** **new** **Rectangle**(res);

}

**@Override**

**public** IShape **rot**(**double** phi) {

// Создание массива вершин повернутого прямоугольника

Point2D[] res = **new** Point2D[n];

**for** (**int** i = **0**; i < n; i++) {

res[i] = p[i].rot(phi);

}

// Создание нового прямоугольника с измененными координатами вершин

**return** **new** **Rectangle**(res);

}

**@Override**

**public** IShape **symAxis**(**int** i) {

// Создание массива вершин отраженного прямоугольника

Point2D[] res = **new** Point2D[n];

**for** (**int** j = **0**; j < n; j++) {

res[j] = (Point2D) p[j].symAxis(i);

}

// Создание нового прямоугольника с измененными координатами вершин

**return** **new** **Rectangle**(res);

}

**public** Document **toDocument**() {

Document document = **new** Document();

document.append("type", "Rectangle");

document.append("n", n);

Document[] points = **new** Document[n];

**for** (**int** i = **0**; i < n; i++) {

points[i] = p[i].toDocument();

}

document.append("points", points);

**return** document;

}

}

## **Segment**

**package** org.figures.figures;

**import** **lombok.ToString**;

**import** **org.bson.Document**;

**import** **org.figures.abstractclasses.OpenFigure**;

**import** **org.figures.figures.points.Point2D**;

**import** **org.figures.interfaces.IShape**;

**import** **lombok.Getter**;

**import** **lombok.Setter**;

**@ToString**

**public** **class** **Segment** **extends** OpenFigure {

**@Getter**

**@Setter**

**private** Point2D start; // начальная точка сегмента

**@Getter**

**@Setter**

**private** Point2D finish; // конечная точка сегмента

// Конструктор, принимающий начальную и конечную точки

**public** **Segment**(Point2D s, Point2D f) {

start = s;

finish = f;

}

// Метод, вычисляющий длину сегмента

**@Override**

**public** **double** **length**() {

**double** dx = finish.getX(**0**) - start.getX(**0**);

**double** dy = finish.getX(**1**) - start.getX(**1**);

**return** Math.sqrt(dx \* dx + dy \* dy);

}

// Метод сдвига (перемещения) отрезка на заданный вектор

**@Override**

**public** IShape **shift**(Point2D a) {

// Создание новых точек с координатами, сдвинутыми на заданный вектор

Point2D newStart = **new** Point2D(**new** **double**[]{start.getX(**0**) + a.getX(**0**), start.getX(**1**) + a.getX(**1**)});

Point2D newFinish = **new** Point2D(**new** **double**[]{finish.getX(**0**) + a.getX(**0**), finish.getX(**1**) + a.getX(**1**)});

// Возвращение нового отрезка с сдвинутыми координатами точек

**return** **new** **Segment**(newStart, newFinish);

}

// Метод поворота отрезка на заданный угол

**@Override**

**public** IShape **rot**(**double** phi) {

// Создание новых точек, полученных путем поворота исходных точек на заданный угол

Point2D newStart = start.rot(phi);

Point2D newFinish = finish.rot(phi);

// Возвращение нового отрезка с повернутыми координатами точек

**return** **new** **Segment**(newStart, newFinish);

}

// Метод симметрии относительно заданной оси

**@Override**

**public** IShape **symAxis**(**int** i) {

// Создание новых точек, полученных путем симметрии исходных точек относительно заданной оси

Point2D newStart = (Point2D) start.symAxis(i);

Point2D newFinish = (Point2D) finish.symAxis(i);

// Возвращение нового отрезка с симметричными координатами точек

**return** **new** **Segment**(newStart, newFinish);

}

**@Override**

**public** **boolean** **cross**(IShape i) {

**if** (!(i **instanceof** Segment)) {

**throw** **new** **IllegalArgumentException**("Аргумент должен быть экземпляром класса Classes.Segment");

}

// Приведение аргумента i к типу Classes.Segment.

Segment other = (Segment) i;

// Получение координат начала и конца отрезков.

Point2D a = start;

Point2D b = finish;

Point2D c = other.start;

Point2D d = other.finish;

// Проверка на совпадение координат начала и конца отрезков.

**if** (a.getX(**0**) == c.getX(**0**) && a.getX(**1**) == c.getX(**1**) && b.getX(**0**) == d.getX(**0**) && b.getX(**1**) == d.getX(**1**))

**return** **true**;

// Расчет значений переменных для дальнейших проверок.

**double** ua = (d.getX(**0**) - c.getX(**0**)) \* (a.getX(**1**) - c.getX(**1**)) - (d.getX(**1**) - c.getX(**1**)) \* (a.getX(**0**) - c.getX(**0**));

**double** ub = (b.getX(**0**) - a.getX(**0**)) \* (a.getX(**1**) - c.getX(**1**)) - (b.getX(**1**) - a.getX(**1**)) \* (a.getX(**0**) - c.getX(**0**));

**double** denom = (d.getX(**1**) - c.getX(**1**)) \* (b.getX(**0**) - a.getX(**0**)) - (d.getX(**0**) - c.getX(**0**)) \* (b.getX(**1**) - a.getX(**1**));

// Проверка на то, что отрезки пересекаются.

**if** (denom == **0**) {

**return** **false**;

}

ua /= denom;

ub /= denom;

**return** ua >= **0** && ua <= **1** && ub >= **0** && ub <= **1**;

}

**public** Document **toDocument**() {

Document document = **new** Document();

document.append("type", "Segment");

document.append("start", start.toDocument());

document.append("finish", finish.toDocument());

**return** document;

}

**public** **double** **square**(){

**return** **0**;

}

}

## **TGon**

**package** org.figures.figures;

**import** **org.bson.Document**;

**import** **org.figures.figures.points.Point2D**;

**import** **org.figures.interfaces.IShape**;

**public** **class** **TGon** **extends** NGon {

**public** **TGon**(Point2D[] p) {

**super**(p);

// Проверяем, что все три точки не лежат на одной прямой

**if** ((p[**1**].getX(**1**) - p[**0**].getX(**1**)) \* (p[**2**].getX(**0**) - p[**1**].getX(**0**)) == (p[**2**].getX(**1**) - p[**1**].getX(**1**)) \* (p[**1**].getX(**0**) - p[**0**].getX(**0**))) {

**throw** **new** **IllegalArgumentException**("Введите корректные координаты для треугольника!");

}

}

**@Override**

**public** **double** **square**() {

// Вычисление длин сторон треугольника

**double** a = Math.sqrt(Math.pow(p[**1**].getX(**0**) - p[**0**].getX(**0**), **2**) + Math.pow(p[**1**].getX(**1**) - p[**0**].getX(**1**), **2**));

**double** b = Math.sqrt(Math.pow(p[**2**].getX(**0**) - p[**1**].getX(**0**), **2**) + Math.pow(p[**2**].getX(**1**) - p[**1**].getX(**1**), **2**));

**double** c = Math.sqrt(Math.pow(p[**0**].getX(**0**) - p[**2**].getX(**0**), **2**) + Math.pow(p[**0**].getX(**1**) - p[**2**].getX(**1**), **2**));

// Вычисление полупериметра

**double** pr = (a + b + c) / **2**;

// Вычисление площади по формуле Герона

**return** Math.sqrt(pr \* (pr - a) \* (pr - b) \* (pr - c));

}

**@Override**

**public** IShape **shift**(Point2D a) {

Point2D[] res = **new** Point2D[n];

// Сдвиг каждой точки на заданный вектор

**for** (**int** i = **0**; i < n; i++) {

res[i] = **new** Point2D(**new** **double**[]{p[i].getX(**0**) + a.getX(**0**), p[i].getX(**1**) + a.getX(**1**)});

}

**return** **new** **TGon**(res);

}

**@Override**

**public** IShape **rot**(**double** phi) {

Point2D[] res = **new** Point2D[n];

// Поворот каждой точки на заданный угол

**for** (**int** i = **0**; i < n; i++) {

res[i] = p[i].rot(phi);

}

**return** **new** **TGon**(res);

}

**@Override**

**public** IShape **symAxis**(**int** i) {

Point2D[] res = **new** Point2D[n];

// Отражение каждой точки относительно заданной оси симметрии

**for** (**int** j = **0**; j < n; j++) {

res[j] = (Point2D) p[j].symAxis(i);

}

**return** **new** **TGon**(res);

}

**public** Document **toDocument**() {

Document document = **new** Document();

document.append("type", "TGon");

document.append("n", n);

Document[] points = **new** Document[n];

**for** (**int** i = **0**; i < n; i++) {

points[i] = p[i].toDocument();

}

document.append("points", points);

**return** document;

}

}

## **Trapeze**

**package** org.figures.figures;

**import** **org.bson.Document**;

**import** **org.figures.figures.points.Point2D**;

**import** **org.figures.interfaces.IShape**;

**public** **class** **Trapeze** **extends** QGon {

**public** **Trapeze**(Point2D[] p) {

**super**(p);

}

**@Override**

**public** **double** **square**() {

**double** k1 = (p[**2**].getX(**1**) - p[**0**].getX(**1**)) / (p[**2**].getX(**0**) - p[**0**].getX(**0**));

**double** k2 = (p[**3**].getX(**1**) - p[**1**].getX(**1**)) / (p[**3**].getX(**0**) - p[**1**].getX(**0**));

**double** phi = Math.atan((k2 - k1) / (**1** + k1 \* k2));

**double** a = Math.sqrt(Math.pow(p[**2**].getX(**0**) - p[**0**].getX(**0**), **2**) + Math.pow(p[**2**].getX(**1**) - p[**0**].getX(**1**), **2**));

**double** b = Math.sqrt(Math.pow(p[**3**].getX(**0**) - p[**1**].getX(**0**), **2**) + Math.pow(p[**3**].getX(**1**) - p[**1**].getX(**1**), **2**));

**return** **0.5** \* a \* b \* Math.abs(Math.sin(phi));

}

/\*

public boolean isTrapezoid() {

boolean parallel = false;

boolean horizontal = false;

for (int i = 0; i < 4; i++) {

int j = (i + 1) % 4;

int k = (i + 2) % 4;

int l = (i + 3) % 4;

if (p[i].getX(1) == p[j].getX(1) && p[k].getX(1) == p[l].getX(1) && p[i].getX(1) == p[k].getX(1)) {

horizontal = true;

if (p[i].getX(0) < p[j].getX(0)) {

if (p[k].getX(0) > p[l].getX(0) && p[k].getX(0) > p[i].getX(0) && p[l].getX(0) < p[j].getX(0)) {

parallel = true;

}

} else {

if (p[l].getX(0) > p[k].getX(0) && p[l].getX(0) > p[j].getX(0) && p[k].getX(0) < p[i].getX(0)) {

parallel = true;

}

}

}

}

return parallel && horizontal;

}

\*/

**@Override**

**public** IShape **shift**(Point2D a) {

Point2D[] res = **new** Point2D[n];

**for** (**int** i = **0**; i < n; i++) {

res[i] = **new** Point2D(**new** **double**[]{p[i].getX(**0**) + a.getX(**0**), p[i].getX(**1**) + a.getX(**1**)});

}

**return** **new** **Trapeze**(res);

}

**@Override**

**public** IShape **rot**(**double** phi) {

Point2D[] res = **new** Point2D[n];

**for** (**int** i = **0**; i < n; i++) {

res[i] = p[i].rot(phi);

}

**return** **new** **Trapeze**(res);

}

**@Override**

**public** IShape **symAxis**(**int** i) {

Point2D[] res = **new** Point2D[n];

**for** (**int** j = **0**; j < n; j++) {

res[j] = (Point2D) p[j].symAxis(i);

}

**return** **new** **Trapeze**(res);

}

**public** Document **toDocument**() {

Document document = **new** Document();

document.append("type", "Trapeze");

document.append("n", n);

Document[] points = **new** Document[n];

**for** (**int** i = **0**; i < n; i++) {

points[i] = p[i].toDocument();

}

document.append("points", points);

**return** document;

}

}

## **Figures**

**package** org.figures;

**import** **org.figures.figures.\***;

**import** **org.figures.figures.points.Point2D**;

**import** **org.figures.interfaces.IShape**;

**import** **java.io.BufferedReader**;

**import** **java.io.FileReader**;

**import** **java.io.IOException**;

**import** **java.nio.charset.StandardCharsets**;

**import** **java.util.ArrayList**;

**import** **java.util.List**;

**import** **java.util.Locale**;

**public** **class** **Figures** {

**private** **final** List<IShape> list = **new** ArrayList<>();

**private** **final** List<IShape> list1 = **new** ArrayList<>();

**private** **final** List<IShape> list2 = **new** ArrayList<>();

**public** **Figures**() {

// Настройка локали

Locale.setDefault(Locale.US);

Figures();

}

**public** **void** **Figures**() {

**int** k1 = **0**;

**int** figures\_count = **0**;

**try** {

BufferedReader reader = **new** BufferedReader(**new** FileReader("files/file.txt", StandardCharsets.UTF\_8));

String n = reader.readLine();

**if** (n != **null**) {

figures\_count = Integer.parseInt(n);

} **else** {

System.out.println("В вашем файле не указано количество фигур!");

}

String line;

**while** ((line = reader.readLine()) != **null**) {

String[] values = line.split(" ");

String figure = values[**0**];

**switch** (figure) {

**case** "Circle":

list.add(**new** Circle(**new** Point2D(**new** **double**[]{Double.parseDouble(values[**1**]), Double.parseDouble(values[**2**])}), Double.parseDouble(values[**3**])));

**break**;

**case** "Segment":

list.add(**new** Segment(**new** Point2D(**new** **double**[]{Double.parseDouble(values[**1**]), Double.parseDouble(values[**2**])}),

**new** **Point2D**(**new** **double**[]{Double.parseDouble(values[**3**]), Double.parseDouble(values[**4**])})));

**break**;

**case** "Polyline":

**case** "NGon":

**case** "QGon":

**case** "TGon":

**case** "Trapeze":

**case** "Rectangle":

**int** k = Integer.parseInt(values[**1**]);

Point2D[] pr = **new** Point2D[k];

**for** (**int** i = **0**, j = **2**; i < k; i++, j += **2**) {

pr[i] = **new** Point2D(**new** **double**[]{Double.parseDouble(values[j]), Double.parseDouble(values[j + **1**])});

}

**if** (figure.equals("Polyline")) list.add(**new** Polyline(pr));

**if** (figure.equals("NGon")) list.add(**new** NGon(pr));

**if** (figure.equals("QGon")) list.add(**new** QGon(pr));

**if** (figure.equals("TGon")) list.add(**new** TGon(pr));

**if** (figure.equals("Trapeze")) list.add(**new** Trapeze(pr));

**if** (figure.equals("Rectangle")) list.add(**new** Rectangle(pr));

**break**;

**default**:

System.out.println("Введён неверный тип фигуры: " + figure);

**break**;

}

}

reader.close();

} **catch** (IOException ex) {

System.out.println("Ошибка чтения файла: " + ex.getMessage());

}

**double** sum\_square = **0**;

**double** sum\_length = **0**;

**for** (IShape v : list) {

//String typeName = v.getClass().getSimpleName();

//System.out.print(typeName + ": ");

//System.out.print(v.toString());

//System.out.print("Длина: " + Math.round(v.length() \* 100.0) / 100.0);

sum\_length += v.length();

//System.out.print("Площадь: " + Math.round(v.square() \* 100.0) / 100.0);

sum\_square += v.square();

}

System.out.println("Суммарная площадь: " + Math.round(sum\_square \* **100.0**) / **100.0**);

System.out.println("Суммарная длина: " + Math.round(sum\_length \* **100.0**) / **100.0**);

System.out.println("Средняя площадь: " + Math.round((sum\_square / figures\_count) \* **100.0**) / **100.0**);

System.out.println();

**try** {

BufferedReader reader = **new** BufferedReader(**new** FileReader("files/file1.txt", StandardCharsets.UTF\_8));

String line;

**int** q = **0**;

**while** ((line = reader.readLine()) != **null**) {

String[] values = line.split(" ");

String figure = list.get(q).getClass().getSimpleName();

**switch** (figure) {

**case** "Circle":

list1.add(**new** Circle(**new** Point2D(**new** **double**[]{Double.parseDouble(values[**0**]), Double.parseDouble(values[**1**])}),

Double.parseDouble(values[**2**])));

**break**;

**case** "Segment":

list1.add(**new** Segment(**new** Point2D(**new** **double**[]{Double.parseDouble(values[**0**]), Double.parseDouble(values[**1**])}),

**new** **Point2D**(**new** **double**[]{Double.parseDouble(values[**2**]), Double.parseDouble(values[**3**])})));

**break**;

**case** "Polyline":

**case** "NGon":

**case** "QGon":

**case** "TGon":

**case** "Trapeze":

**case** "Rectangle":

**int** k = Integer.parseInt(values[**0**]);

Point2D[] pr = **new** Point2D[k];

**for** (**int** i = **0**, j = **1**; i < k; i++, j += **2**) {

pr[i] = **new** Point2D(**new** **double**[]{Double.parseDouble(values[j]), Double.parseDouble(values[j + **1**])});

}

**if** (figure.equals("Polyline")) list1.add(**new** Polyline(pr));

**if** (figure.equals("NGon")) list1.add(**new** NGon(pr));

**if** (figure.equals("QGon")) list1.add(**new** QGon(pr));

**if** (figure.equals("TGon")) list1.add(**new** TGon(pr));

**if** (figure.equals("Trapeze")) list1.add(**new** Trapeze(pr));

**if** (figure.equals("Rectangle")) list1.add(**new** Rectangle(pr));

**break**;

**default**:

System.out.println("Введён неверный тип фигуры: " + figure);

**break**;

}

q++;

}

reader.close();

} **catch** (IOException ex) {

System.out.println("Ошибка чтения файла: " + ex.getMessage());

}

**for** (**int** i = **0**; i < figures\_count; i++) {

String typeName = list.get(i).getClass().getSimpleName();

System.out.print("Пересекаются ли " + typeName + ": ");

**if** (list.get(i).cross(list1.get(i))) {

System.out.println("да");

} **else** {

System.out.println("нет");

}

}

**try** {

BufferedReader reader = **new** BufferedReader(**new** FileReader("files/file2.txt", StandardCharsets.UTF\_8));

String line;

**int** w = **0**;

**while** ((line = reader.readLine()) != **null**) {

String[] values = line.split(" ");

String move = values[**0**];

**switch** (move) {

**case** "rot":

list2.add(list1.get(w).rot(Double.parseDouble(values[**1**])));

**break**;

**case** "symAxis":

list2.add(list1.get(w).symAxis(Integer.parseInt(values[**1**])));

**break**;

**case** "shift":

list2.add(list1.get(w).shift(**new** Point2D(**new** **double**[]{Double.parseDouble(values[**1**]), Double.parseDouble(values[**2**])})));

**break**;

**default**:

System.out.println("Введён неверный тип фигуры: " + move);

**break**;

}

w++;

}

reader.close();

} **catch** (IOException ex) {

System.out.println("Ошибка чтения файла: " + ex.getMessage());

}

System.out.println();

System.out.print("После движения фигур: ");

System.out.println();

**for** (**int** i = **0**; i < figures\_count; i++) {

String typeName = list.get(i).getClass().getSimpleName();

System.out.print("Пересекаются ли " + typeName + ": ");

**if** (list.get(i).cross(list2.get(i))) {

System.out.println("да");

} **else** {

System.out.println("нет");

}

}

}

}

## **PointTest**

**package** org.figures.figures.points;

**import** **org.bson.Document**;

**import** **org.junit.jupiter.api.Test**;

**import** **static** org.junit.jupiter.api.Assertions.\*;

**class** **PointTest** {

**@Test**

**void** **testConstructorWithInvalidDimensions**() {

// Подготовка данных для теста

**int** dim = **3**;

**double**[] invalidX = {**1.0**, **2.0**}; // Массив с неправильной размерностью

// Проверка, что конструктор вызывает исключение с правильным сообщением

assertThrows(IllegalArgumentException.class, () -> **new** Point(dim, invalidX),

"Размерность массива не совпадает с заданной размерностью пространства.");

}

**@Test**

**void** **testGetXWithInvalidIndex**() {

// Подготовка данных для теста

**int** dim = **3**;

**double**[] coordinates = {**1.0**, **2.0**, **3.0**};

Point point = **new** Point(dim, coordinates);

assertThrows(IllegalArgumentException.class, () -> point.getX(dim),

"Неверный индекс координаты.");

}

**@Test**

**void** **testSetXWithInvalidDimensions**() {

**int** dim = **3**;

**double**[] validX = {**1.0**, **2.0**, **3.0**};

**double**[] invalidX = {**4.0**, **5.0**};

Point point = **new** Point(dim, validX);

assertThrows(IllegalArgumentException.class, () -> point.setX(invalidX),

"Размерность массива не совпадает с заданной размерностью пространства.");

}

**@Test**

**void** **testSetXWithInvalidIndex**() {

**int** dim = **3**;

**double**[] coordinates = {**1.0**, **2.0**, **3.0**};

Point point = **new** Point(dim, coordinates);

assertThrows(IllegalArgumentException.class, () -> point.setX(**4.0**, dim),

"Неверный индекс координаты.");

}

**@Test**

**void** **testAddWithMismatchedDimensions**() {

**int** dim1 = **3**;

**int** dim2 = **2**;

**double**[] coordinates1 = {**1.0**, **2.0**, **3.0**};

**double**[] coordinates2 = {**4.0**, **5.0**};

Point point1 = **new** Point(dim1, coordinates1);

Point point2 = **new** Point(dim2, coordinates2);

assertThrows(IllegalArgumentException.class, () -> Point.add(point1, point2),

"Размерности точек не совпадают.");

}

**@Test**

**void** **testInstanceAddWithMismatchedDimensions**() {

**int** dim1 = **3**;

**int** dim2 = **2**;

**double**[] coordinates1 = {**1.0**, **2.0**, **3.0**};

**double**[] coordinates2 = {**4.0**, **5.0**};

Point point1 = **new** Point(dim1, coordinates1);

Point point2 = **new** Point(dim2, coordinates2);

assertThrows(IllegalArgumentException.class, () -> point1.add(point2),

"Размерности точек не совпадают.");

}

**@Test**

**void** **testStaticSubWithMismatchedDimensions**() {

**int** dim1 = **3**;

**int** dim2 = **2**;

**double**[] coordinates1 = {**1.0**, **2.0**, **3.0**};

**double**[] coordinates2 = {**4.0**, **5.0**};

Point point1 = **new** Point(dim1, coordinates1);

Point point2 = **new** Point(dim2, coordinates2);

assertThrows(IllegalArgumentException.class, () -> Point.sub(point1, point2),

"Размерности точек не совпадают.");

}

**@Test**

**void** **testNonStaticSubWithMismatchedDimensions**() {

**int** dim1 = **3**;

**int** dim2 = **2**;

**double**[] coordinates1 = {**1.0**, **2.0**, **3.0**};

**double**[] coordinates2 = {**4.0**, **5.0**};

Point point1 = **new** Point(dim1, coordinates1);

Point point2 = **new** Point(dim2, coordinates2);

assertThrows(IllegalArgumentException.class, () -> point1.sub(point2),

"Размерности точек не совпадают.");

}

**@Test**

**void** **testStaticMultWithMismatchedDimensions**() {

**int** dim1 = **3**;

**int** dim2 = **2**;

**double**[] coordinates1 = {**1.0**, **2.0**, **3.0**};

**double**[] coordinates2 = {**4.0**, **5.0**};

Point point1 = **new** Point(dim1, coordinates1);

Point point2 = **new** Point(dim2, coordinates2);

assertThrows(IllegalArgumentException.class, () -> Point.mult(point1, point2),

"Размерности точек не совпадают.");

}

**@Test**

**void** **testMultWithMismatchedDimensions**() {

**int** dim1 = **3**;

**int** dim2 = **2**;

**double**[] coordinates1 = {**1.0**, **2.0**, **3.0**};

**double**[] coordinates2 = {**4.0**, **5.0**};

Point point1 = **new** Point(dim1, coordinates1);

Point point2 = **new** Point(dim2, coordinates2);

assertThrows(IllegalArgumentException.class, () -> point1.mult(point2),

"Размерности точек не совпадают.");

}

**@Test**

**void** **testGetXWithInvalidIndexBelowZero**() {

**int** dim = **3**;

**double**[] coordinates = {**1.0**, **2.0**, **3.0**};

Point point = **new** Point(dim, coordinates);

**int** invalidIndex = -**1**; // Неверный индекс (меньше 0)

IllegalArgumentException thrownException = assertThrows(IllegalArgumentException.class, () -> point.getX(invalidIndex),

"Неверный индекс координаты.");

assertEquals("Неверный индекс координаты.", thrownException.getMessage(), "Сообщение исключения верное");

}

**@Test**

**void** **testSetXWithNegativeIndex**() {

**int** dim = **3**;

**double**[] coordinates = {**1.0**, **2.0**, **3.0**};

Point point = **new** Point(dim, coordinates);

IllegalArgumentException thrownException = assertThrows(IllegalArgumentException.class, () -> point.setX(**0.0**, -**1**),

"Неверный индекс координаты.");

assertEquals("Неверный индекс координаты.", thrownException.getMessage(), "Сообщение исключения верное");

}

**@Test**

**void** **testGetXWithInvalidIndexAboveDim**() {

**int** dim = **3**;

**double**[] coordinates = {**1.0**, **2.0**, **3.0**};

Point point = **new** Point(dim, coordinates);

IllegalArgumentException thrownException = assertThrows(IllegalArgumentException.class, () -> point.getX(dim),

"Неверный индекс координаты.");

assertEquals("Неверный индекс координаты.", thrownException.getMessage(), "Сообщение исключения верное");

}

**@Test**

**void** **getDim**() {

Point point = **new** Point(**3**);

assertEquals(**3**, point.getDim(), "Dimensions are equal");

}

**@Test**

**void** **getX**() {

**double**[] coordinates = {**1.0**, **2.0**, **3.0**};

Point point = **new** Point(**3**, coordinates);

assertArrayEquals(coordinates, point.getX(), **0.001**, "Coordinates are equal");

}

**@Test**

**void** **testGetX**() {

**double**[] coordinates = {**1.0**, **2.0**, **3.0**};

Point point = **new** Point(**3**, coordinates);

assertAll("coordinates",

() -> assertEquals(**1.0**, point.getX(**0**), **0.001**),

() -> assertEquals(**2.0**, point.getX(**1**), **0.001**),

() -> assertEquals(**3.0**, point.getX(**2**), **0.001**)

);

}

**@Test**

**void** **setX**() {

Point point = **new** Point(**3**);

**double**[] newCoordinates = {**4.0**, **5.0**, **6.0**};

point.setX(newCoordinates);

assertArrayEquals(newCoordinates, point.getX(), **0.001**, "Coordinates are equal");

}

**@Test**

**void** **testSetX**() {

**double**[] coordinates = {**1.0**, **2.0**, **3.0**};

Point point = **new** Point(**3**, coordinates);

point.setX(**4.0**, **1**);

assertEquals(**4.0**, point.getX(**1**), **0.001**, "Coordinates are equal");

}

**@Test**

**void** **abs**() {

**double**[] coordinates = {**3.0**, **4.0**};

Point point = **new** Point(**2**, coordinates);

assertEquals(**5.0**, point.abs(), **0.001**);

}

**@Test**

**void** **add**() {

**double**[] coordinates1 = {**1.0**, **2.0**};

**double**[] coordinates2 = {**3.0**, **4.0**};

Point point1 = **new** Point(**2**, coordinates1);

Point point2 = **new** Point(**2**, coordinates2);

Point result = Point.add(point1, point2);

**double**[] expected = {**4.0**, **6.0**};

assertArrayEquals(expected, result.getX(), **0.001**);

}

**@Test**

**void** **testAdd**() {

**double**[] coordinates1 = {**1.0**, **2.0**};

**double**[] coordinates2 = {**3.0**, **4.0**};

Point point1 = **new** Point(**2**, coordinates1);

Point point2 = **new** Point(**2**, coordinates2);

Point result = point1.add(point2);

**double**[] expected = {**4.0**, **6.0**};

assertArrayEquals(expected, result.getX(), **0.001**);

}

**@Test**

**void** **sub**() {

**double**[] coordinates1 = {**1.0**, **2.0**};

**double**[] coordinates2 = {**3.0**, **4.0**};

Point point1 = **new** Point(**2**, coordinates1);

Point point2 = **new** Point(**2**, coordinates2);

Point result = Point.sub(point1, point2);

**double**[] expected = {-**2.0**, -**2.0**};

assertArrayEquals(expected, result.getX(), **0.001**);

}

**@Test**

**void** **testSub**() {

**double**[] coordinates1 = {**1.0**, **2.0**};

**double**[] coordinates2 = {**3.0**, **4.0**};

Point point1 = **new** Point(**2**, coordinates1);

Point point2 = **new** Point(**2**, coordinates2);

Point result = point1.sub(point2);

**double**[] expected = {-**2.0**, -**2.0**};

assertArrayEquals(expected, result.getX(), **0.001**);

}

**@Test**

**void** **mult**() {

**double**[] coordinates = {**1.0**, **2.0**, **3.0**};

Point point = **new** Point(**3**, coordinates);

Point result = Point.mult(point, **2.0**);

**double**[] expected = {**2.0**, **4.0**, **6.0**};

assertArrayEquals(expected, result.getX(), **0.001**);

}

**@Test**

**void** **testMult**() {

**double**[] coordinates = {**1.0**, **2.0**, **3.0**};

Point point = **new** Point(**3**, coordinates);

Point result = point.mult(**2.0**);

**double**[] expected = {**2.0**, **4.0**, **6.0**};

assertArrayEquals(expected, result.getX(), **0.001**);

}

**@Test**

**void** **testMult1**() {

**double**[] coordinates1 = {**1.0**, **2.0**};

**double**[] coordinates2 = {**3.0**, **4.0**};

Point point1 = **new** Point(**2**, coordinates1);

Point point2 = **new** Point(**2**, coordinates2);

**double** result = Point.mult(point1, point2);

assertEquals(**11.0**, result, **0.001**);

}

**@Test**

**void** **testMult2**() {

**double**[] coordinates1 = {**1.0**, **2.0**};

**double**[] coordinates2 = {**3.0**, **4.0**};

Point point1 = **new** Point(**2**, coordinates1);

Point point2 = **new** Point(**2**, coordinates2);

**double** result = point1.mult(point2);

assertEquals(**11.0**, result, **0.001**);

}

**@Test**

**void** **testSymAxisWithInvalidAxisIndex\_Static**() {

**int** dim = **3**;

**double**[] coordinates = {**1.0**, **2.0**, **3.0**};

Point point = **new** Point(dim, coordinates);

IllegalArgumentException thrownException = assertThrows(IllegalArgumentException.class, () -> Point.symAxis(point, **5**),

"Неверный номер оси.");

assertEquals("Неверный номер оси.", thrownException.getMessage(), "Сообщение исключения верное");

}

**@Test**

**void** **testSymAxisWithInvalidAxisIndex\_Instance**() {

**int** dim = **3**;

**double**[] coordinates = {**1.0**, **2.0**, **3.0**};

Point point = **new** Point(dim, coordinates);

IllegalArgumentException thrownException = assertThrows(IllegalArgumentException.class, () -> point.symAxis(**5**),

"Неверный номер оси.");

assertEquals("Неверный номер оси.", thrownException.getMessage(), "Сообщение исключения верное");

}

**@Test**

**void** **testSymAxisWithInvalidPositiveAxisIndex\_Static**() {

**int** dim = **3**;

**double**[] coordinates = {**1.0**, **2.0**, **3.0**};

Point point = **new** Point(dim, coordinates);

IllegalArgumentException thrownException = assertThrows(IllegalArgumentException.class, () -> Point.symAxis(point, -**1**),

"Неверный номер оси.");

assertEquals("Неверный номер оси.", thrownException.getMessage(), "Сообщение исключения верное");

}

**@Test**

**void** **testSymAxisWithInvalidPositiveAxisIndex\_Instance**() {

**int** dim = **3**;

**double**[] coordinates = {**1.0**, **2.0**, **3.0**};

Point point = **new** Point(dim, coordinates);

IllegalArgumentException thrownException = assertThrows(IllegalArgumentException.class, () -> point.symAxis(-**1**),

"Неверный номер оси.");

assertEquals("Неверный номер оси.", thrownException.getMessage(), "Сообщение исключения верное");

}

**@Test**

**void** **symAxis**() {

**double**[] coordinates = {**1.0**, **2.0**, **3.0**};

Point point = **new** Point(**3**, coordinates);

Point result = Point.symAxis(point, **1**);

**double**[] expected = {-**1.0**, **2.0**, -**3.0**};

assertArrayEquals(expected, result.getX(), **0.001**);

}

**@Test**

**void** **testSymAxis**() {

**double**[] coordinates = {**1.0**, **2.0**, **3.0**};

Point point = **new** Point(**3**, coordinates);

Point result = point.symAxis(**1**);

**double**[] expected = {-**1.0**, **2.0**, -**3.0**};

assertArrayEquals(expected, result.getX(), **0.001**);

}

**@Test**

**void** **testToString**() {

**double**[] coordinates = {**1.0**, **2.0**, **3.0**};

Point point = **new** Point(**3**, coordinates);

String expectedString = "Point(dim=3, x=[1.0, 2.0, 3.0])";

assertEquals(expectedString, point.toString());

}

**@Test**

**public** **void** **testToDocument**() {

// Создаем экземпляр класса Point с одним параметром

Point point1 = **new** Point(**2**);

Document document1 = point1.toDocument();

// Проверяем, что документ создан корректно

assertEquals("Point", document1.getString("type"));

assertEquals(**2**, document1.getInteger("dim"));

assertNotNull(document1.get("x")); // Проверяем, что поле x не равно null

// Создаем экземпляр класса Point с двумя параметрами

Point point2 = **new** Point(**3**, **new** **double**[]{**1.0**, **2.0**, **3.0**});

Document document2 = point2.toDocument();

// Проверяем, что документ создан корректно

assertEquals("Point", document2.getString("type"));

assertEquals(**3**, document2.getInteger("dim"));

assertNotNull(document2.get("x")); // Проверяем, что поле x не равно null

}

}

## **Point2DTest**

**package** org.figures.figures.points;

**import** **org.bson.Document**;

**import** **org.junit.jupiter.api.Test**;

**import** **static** org.junit.jupiter.api.Assertions.\*;

**class** **Point2DTest** {

**@Test**

**void** **testPoint2DConstructorWithoutParameters**() {

// Создаем точку с использованием конструктора без параметров

Point2D point2D = **new** Point2D();

// Проверяем, что размерность точки равна 2

assertEquals(**2**, point2D.getDim(), "Размерность точки должна быть равна 2");

// Проверяем, что массив координат инициализирован и имеет длину 2

assertNotNull(point2D.getX(), "Массив координат не должен быть null");

assertEquals(**2**, point2D.getX().length, "Длина массива координат должна быть равна 2");

}

**@Test**

**void** **testSymAxis**() {

// Создаем точку (1.0, 2.0)

Point2D point2D = **new** Point2D(**new** **double**[]{**1.0**, **2.0**});

// Тестируем симметрию относительно оси i = 0

Point symAxisResult1 = point2D.symAxis(**0**);

assertTrue(symAxisResult1 **instanceof** Point2D, "Ожидается объект типа Point2D");

assertArrayEquals(**new** **double**[]{**1.0**, -**2.0**}, symAxisResult1.getX(), **0.001**, "Симметрия относительно оси i=0 не верна");

// Тестируем симметрию относительно оси i = 1

Point symAxisResult2 = point2D.symAxis(**1**);

assertTrue(symAxisResult2 **instanceof** Point2D, "Ожидается объект типа Point2D");

assertArrayEquals(**new** **double**[]{-**1.0**, **2.0**}, symAxisResult2.getX(), **0.001**, "Симметрия относительно оси i=1 не верна");

// Тестируем выброс исключения при неверном номере оси

assertThrows(IllegalArgumentException.class, () -> point2D.symAxis(**2**), "Ожидается выброс IllegalArgumentException");

// Тестируем выброс исключения при i < 0

assertThrows(IllegalArgumentException.class, () -> point2D.symAxis(-**1**), "Ожидается выброс IllegalArgumentException");

}

**@Test**

**void** **rot**() {

**double**[] coordinates = {**1.0**, **0.0**};

Point2D point = **new** Point2D(coordinates);

Point2D result = Point2D.rot(point, Math.PI / **2**);

**double**[] expected = {**0.0**, **1.0**};

assertArrayEquals(expected, result.getX(), **0.001**);

}

**@Test**

**void** **testRot**() {

**double**[] coordinates = {**1.0**, **0.0**};

Point2D point = **new** Point2D(coordinates);

Point2D result = point.rot(Math.PI / **2**);

**double**[] expected = {**0.0**, **1.0**};

assertArrayEquals(expected, result.getX(), **0.001**);

}

**@Test**

**void** **symAxis**() {

**double**[] coordinates = {**1.0**, **2.0**};

Point2D point = **new** Point2D(coordinates);

Point result = point.symAxis(**0**);

**double**[] expected = {**1.0**, -**2.0**};

assertArrayEquals(expected, result.getX(), **0.001**);

}

**@Test**

**public** **void** **testToDocument**() {

// Создаем экземпляр класса Point2D с конструктором без параметров

Point2D point1 = **new** Point2D();

Document document1 = point1.toDocument();

// Проверяем, что документ создан корректно

assertEquals("Point2D", document1.getString("type"));

assertEquals(**2**, document1.getInteger("dim"));

assertNotNull(document1.get("x")); // Проверяем, что поле x не равно null

// Создаем экземпляр класса Point2D с конструктором, принимающим массив координат

Point2D point2 = **new** Point2D(**new** **double**[]{**1.0**, **2.0**});

Document document2 = point2.toDocument();

// Проверяем, что документ создан корректно

assertEquals("Point2D", document2.getString("type"));

assertEquals(**2**, document2.getInteger("dim"));

assertNotNull(document2.get("x")); // Проверяем, что поле x не равно null

}

}

## **Point3DTest**

**package** org.figures.figures.points;

**import** **org.bson.Document**;

**import** **org.junit.jupiter.api.BeforeAll**;

**import** **org.junit.jupiter.api.Test**;

**import** **static** org.junit.jupiter.api.Assertions.\*;

**class** **Point3DTest** {

**static** Point3D point1;

**static** Point3D point2;

**static** Point3D point3;

**@BeforeAll**

**static** **void** **setUp**(){

**double**[] coordinates1 = {**1.0**, **2.0**, **3.0**};

point1 = **new** Point3D(coordinates1);

**double**[] coordinates2 = {**4.0**, **5.0**, **6.0**};

point2 = **new** Point3D(coordinates2);

**double**[] coordinates3 = {**7.0**, **8.0**, **9.0**};

point3 = **new** Point3D(coordinates3);

}

**@Test**

**void** **testConstructor**() {

Point3D point3D = **new** Point3D();

// Проверяем, что размерность точки 3

assertEquals(**3**, point3D.getDim(), "Размерность точки должна быть 3");

// Проверяем, что все координаты инициализированы нулями

assertArrayEquals(**new** **double**[]{**0.0**, **0.0**, **0.0**}, point3D.getX(), **0.001**, "Координаты должны быть нулевыми");

}

**@Test**

**void** **cross\_prod**() {

Point3D result = Point3D.cross\_prod(point1, point2);

**double**[] expected = {-**3.0**, **6.0**, -**3.0**};

assertArrayEquals(expected, result.getX(), **0.001**);

}

**@Test**

**void** **testCross\_prod**() {

Point3D result = point1.cross\_prod(point2);

**double**[] expected = {-**3.0**, **6.0**, -**3.0**};

assertArrayEquals(expected, result.getX(), **0.001**);

}

**@Test**

**void** **mix\_prod**() {

**double** result = Point3D.mix\_prod(point1, point2, point3);

**double** expected = **0.0**;

assertEquals(expected, result, **0.001**);

}

**@Test**

**void** **testMix\_prod**() {

**double** result = point1.mix\_prod(point2, point3);

**double** expected = **0.0**;

assertEquals(expected, result, **0.001**);

}

**@Test**

**public** **void** **testToDocument**() {

Point3D point1 = **new** Point3D();

Document document1 = point1.toDocument();

assertEquals("Point3D", document1.getString("type"));

assertEquals(**3**, document1.getInteger("dim"));

assertNotNull(document1.get("x"));

Point3D point2 = **new** Point3D(**new** **double**[]{**1.0**, **2.0**, **3.0**});

Document document2 = point2.toDocument();

assertEquals("Point3D", document2.getString("type"));

assertEquals(**3**, document2.getInteger("dim"));

assertNotNull(document2.get("x"));

}

}

## **CircleTest**

**package** org.figures.figures;

**import** **org.bson.Document**;

**import** **org.junit.jupiter.api.Test**;

**import** **org.figures.figures.points.Point2D**;

**import** **org.figures.interfaces.IShape**;

**import** **static** org.junit.jupiter.api.Assertions.\*;

**class** **CircleTest** {

**@Test**

**void** **getP**() {

Point2D center = **new** Point2D(**new** **double**[]{**1.0**, **2.0**});

**double** radius = **3.0**;

Circle circle = **new** Circle(center, radius);

Point2D result = circle.getP();

assertEquals(center, result);

}

**@Test**

**void** **setP**() {

Point2D center = **new** Point2D(**new** **double**[]{**1.0**, **2.0**});

**double** radius = **3.0**;

Circle circle = **new** Circle(center, radius);

Point2D newCenter = **new** Point2D(**new** **double**[]{**4.0**, **5.0**});

circle.setP(newCenter);

Point2D result = circle.getP();

assertEquals(newCenter, result);

}

**@Test**

**void** **getR**() {

Point2D center = **new** Point2D(**new** **double**[]{**1.0**, **2.0**});

**double** radius = **3.0**;

Circle circle = **new** Circle(center, radius);

**double** result = circle.getR();

assertEquals(radius, result, **0.001**);

}

**@Test**

**void** **setR**() {

Point2D center = **new** Point2D(**new** **double**[]{**1.0**, **2.0**});

**double** radius = **3.0**;

Circle circle = **new** Circle(center, radius);

**double** newRadius = **4.0**;

circle.setR(newRadius);

**double** result = circle.getR();

assertEquals(newRadius, result, **0.001**);

}

**@Test**

**void** **square**() {

Point2D center = **new** Point2D(**new** **double**[]{**0.0**, **0.0**});

**double** radius = **2.0**;

Circle circle = **new** Circle(center, radius);

**double** expectedSquare = Math.PI \* radius \* radius;

assertEquals(expectedSquare, circle.square(), **0.001**);

}

**@Test**

**void** **length**() {

Point2D center = **new** Point2D(**new** **double**[]{**0.0**, **0.0**});

**double** radius = **2.0**;

Circle circle = **new** Circle(center, radius);

**double** expectedLength = **2** \* Math.PI \* radius;

assertEquals(expectedLength, circle.length(), **0.001**);

}

**@Test**

**void** **shift**() {

Point2D center = **new** Point2D(**new** **double**[]{**2.0**, **2.0**});

**double** radius = **3.0**;

Circle circle = **new** Circle(center, radius);

Point2D shiftVector = **new** Point2D(**new** **double**[]{**2.0**, **3.0**});

IShape shiftedCircle = circle.shift(shiftVector);

Point2D expectedCenter = **new** Point2D(**new** **double**[]{**4.0**, **5.0**});

assertAll("shiftedCircle",

() -> assertEquals(expectedCenter.getX(**0**), ((Circle) shiftedCircle).getP().getX(**0**), **0.001**),

() -> assertEquals(expectedCenter.getX(**1**), ((Circle) shiftedCircle).getP().getX(**1**), **0.001**)

);

}

**@Test**

**void** **rot**() {

Point2D center = **new** Point2D(**new** **double**[]{**1.0**, **2.0**});

**double** radius = **3.0**;

Circle circle = **new** Circle(center, radius);

**double** angle = Math.PI / **2**;

IShape rotatedCircle = circle.rot(angle);

Point2D expectedCenter = **new** Point2D(**new** **double**[]{-**2.0**, **1.0**});

assertAll("rotatedCircle",

() -> assertEquals(expectedCenter.getX(**0**), ((Circle) rotatedCircle).getP().getX(**0**), **0.001**),

() -> assertEquals(expectedCenter.getX(**1**), ((Circle) rotatedCircle).getP().getX(**1**), **0.001**),

() -> assertEquals(radius, ((Circle) rotatedCircle).getR(), **0.001**)

);

}

**@Test**

**void** **symAxis**() {

Point2D center = **new** Point2D(**new** **double**[]{**1.0**, **2.0**});

**double** radius = **3.0**;

Circle circle = **new** Circle(center, radius);

**int** axis = **0**; // Ось x

IShape symmetricCircle = circle.symAxis(axis);

Point2D expectedCenter = **new** Point2D(**new** **double**[]{**1.0**, -**2.0**});

assertAll("symmetricCircle",

() -> assertEquals(expectedCenter.getX(**0**), ((Circle) symmetricCircle).getP().getX(**0**), **0.001**),

() -> assertEquals(expectedCenter.getX(**1**), ((Circle) symmetricCircle).getP().getX(**1**), **0.001**),

() -> assertEquals(radius, ((Circle) symmetricCircle).getR(), **0.001**)

);

}

**@Test**

**void** **testToString**() {

Point2D center = **new** Point2D(**new** **double**[]{**1.0**, **2.0**});

**double** radius = **3.0**;

Circle circle = **new** Circle(center, radius);

String expectedString = "Circle(r=3.0, p=Point(dim=2, x=[1.0, 2.0]))";

assertEquals(expectedString, circle.toString());

}

**@Test**

**void** **testConstructorWithInvalidRadius**() {

Point2D center = **new** Point2D();

**double** invalidRadius = **0**;

assertThrows(IllegalArgumentException.class, () -> **new** Circle(center, invalidRadius),

"Конструктор Circle должен выбрасывать исключение при недопустимом радиусе");

}

**@Test**

**void** **testCircleCross**() {

Point2D center1 = **new** Point2D(**new** **double**[]{**0**, **0**});

**double** radius1 = **3**;

Circle circle1 = **new** Circle(center1, radius1);

assertThrows(IllegalArgumentException.class, () -> {

circle1.cross(**new** NGon(**new** Point2D[]{**new** Point2D(**new** **double**[]{**1**, **1**}), **new** Point2D(**new** **double**[]{**2**, **2**}), **new** Point2D(**new** **double**[]{**3**, **3**})}));

}, "Метод cross должен выбрасывать IllegalArgumentException при передаче не Circle объекта");

Point2D center2 = **new** Point2D(**new** **double**[]{**4**, **0**});

**double** radius2 = **2**;

Circle circle2 = **new** Circle(center2, radius2);

assertTrue(circle1.cross(circle2), "Метод cross должен возвращать true для пересекающихся окружностей");

Point2D center3 = **new** Point2D(**new** **double**[]{**8**, **0**});

**double** radius3 = **1**;

Circle circle3 = **new** Circle(center3, radius3);

assertFalse(circle1.cross(circle3), "Метод cross должен возвращать false для не пересекающихся окружностей");

Point2D center4 = **new** Point2D(**new** **double**[]{**3**, **0**});

**double** radius4 = **1**;

Circle circle4 = **new** Circle(center4, radius4);

assertTrue(circle1.cross(circle4));

Point2D center5 = **new** Point2D(**new** **double**[]{**0.0**, **0.0**});

**double** radius5 = **2.0**;

Circle circle5 = **new** Circle(center5, radius5);

Point2D center6 = **new** Point2D(**new** **double**[]{**3.0**, **0.0**});

**double** radius6 = **2.0**;

Circle circle6 = **new** Circle(center6, radius6);

assertAll("cross",

() -> assertTrue(circle5.cross(circle6))

);

Point2D center7 = **new** Point2D(**new** **double**[]{**8.0**, **0.0**});

**double** radius7 = **2.0**;

Circle circle7 = **new** Circle(center7, radius7);

assertAll("notCross",

() -> assertFalse(circle5.cross(circle7))

);

Point2D center10 = **new** Point2D(**new** **double**[]{**3**, **3**});

**double** radius10 = **3**;

Circle circle10 = **new** Circle(center10, radius10);

assertTrue(circle1.cross(circle10), "Метод cross должен возвращать true для касающихся окружностей");

Point2D center11 = **new** Point2D(**new** **double**[]{**10**, **10**});

**double** radius11 = **2**;

Circle circle11 = **new** Circle(center11, radius11);

assertFalse(circle1.cross(circle11), "Метод cross должен возвращать false для не пересекающихся и не касающихся окружностей");

// Добавленные проверки

Point2D center12 = **new** Point2D(**new** **double**[]{**5**, **0**});

**double** radius12 = **1**;

Circle circle12 = **new** Circle(center12, radius12);

assertFalse(circle1.cross(circle12), "Метод cross должен возвращать false для res < (r + c.r) && res > Math.abs(r - c.r)");

Point2D center13 = **new** Point2D(**new** **double**[]{**0**, **5**});

**double** radius13 = **1**;

Circle circle13 = **new** Circle(center13, radius13);

assertFalse(circle1.cross(circle13), "Метод cross должен возвращать false для res > (r + c.r) && res < Math.abs(r - c.r)");

Point2D center14 = **new** Point2D(**new** **double**[]{**3**, **0**});

**double** radius14 = **6**;

Circle circle14 = **new** Circle(center14, radius14);

assertFalse(circle1.cross(circle14), "Метод cross должен возвращать true для res < (r + c.r) && res > Math.abs(r - c.r)");

Point2D center15 = **new** Point2D(**new** **double**[]{**0**, **3**});

**double** radius15 = **6**;

Circle circle15 = **new** Circle(center15, radius15);

assertFalse(circle1.cross(circle15), "Метод cross должен возвращать true для res > (r + c.r) && res < Math.abs(r - c.r)");

}

**@Test**

**public** **void** **testToDocument**() {

// Создаем экземпляр класса Point2D для центра окружности

Point2D center = **new** Point2D(**new** **double**[]{**1.0**, **2.0**});

// Создаем экземпляр класса Circle

Circle circle = **new** Circle(center, **3.0**);

Document document = circle.toDocument();

// Проверяем, что документ создан корректно

assertEquals("Circle", document.getString("type"));

assertEquals(**3.0**, document.getDouble("r"));

assertNotNull(document.get("p")); // Проверяем, что поле p не равно null

assertEquals("Point2D", document.get("p", Document.class).getString("type"));

assertEquals(**2**, document.get("p", Document.class).getInteger("dim"));

assertNotNull(document.get("p", Document.class).get("x")); // Проверяем, что поле x внутреннего Point2D не равно null

}

}

## **NGonTest**

**package** org.figures.figures;

**import** **org.bson.Document**;

**import** **org.figures.figures.points.Point**;

**import** **org.figures.interfaces.IShape**;

**import** **org.junit.jupiter.api.Test**;

**import** **org.figures.figures.points.Point2D**;

**import** **static** org.junit.jupiter.api.Assertions.\*;

**class** **NGonTest** {

**@Test**

**void** **getN**() {

Point2D[] vertices = {

**new** **Point2D**(**new** **double**[]{**0.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, **4.0**}),

**new** **Point2D**(**new** **double**[]{**0.0**, **4.0**})

};

NGon ngon = **new** NGon(vertices);

**int** expectedN = **4**; // Ожидаемое количество вершин

assertEquals(expectedN, ngon.getN());

}

**@Test**

**void** **getP**() {

Point2D[] vertices = {

**new** **Point2D**(**new** **double**[]{**0.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, **4.0**}),

**new** **Point2D**(**new** **double**[]{**0.0**, **4.0**})

};

NGon ngon = **new** NGon(vertices);

assertArrayEquals(vertices, ngon.getP());

}

**@Test**

**void** **testGetP**() {

Point2D[] vertices = {

**new** **Point2D**(**new** **double**[]{**0.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, **4.0**}),

**new** **Point2D**(**new** **double**[]{**0.0**, **4.0**})

};

NGon ngon = **new** NGon(vertices);

**int** index = **2**;

Point2D expectedVertex = vertices[index];

assertEquals(expectedVertex, ngon.getP(index));

}

**@Test**

**void** **setP**() {

Point2D[] vertices = {

**new** **Point2D**(**new** **double**[]{**0.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, **4.0**}),

**new** **Point2D**(**new** **double**[]{**0.0**, **4.0**})

};

NGon ngon = **new** NGon(vertices);

Point2D[] newVertices = {

**new** **Point2D**(**new** **double**[]{**1.0**, **1.0**}),

**new** **Point2D**(**new** **double**[]{**5.0**, **1.0**}),

**new** **Point2D**(**new** **double**[]{**5.0**, **5.0**}),

**new** **Point2D**(**new** **double**[]{**1.0**, **5.0**})

};

ngon.setP(newVertices);

assertArrayEquals(newVertices, ngon.getP());

}

**@Test**

**void** **testTGonConstructorWithInvalidCoordinates**() {

Point2D point1 = **new** Point2D(**new** **double**[]{**0**, **0**});

Point2D point2 = **new** Point2D(**new** **double**[]{**1**, **0**});

Point2D point3 = **new** Point2D(**new** **double**[]{**2**, **0**});

assertThrows(IllegalArgumentException.class, () -> **new** TGon(**new** Point2D[]{point1, point2, point3}),

"Конструктор TGon должен выбрасывать исключение при некорректных координатах");

}

**@Test**

**void** **testGetPWithNegativeIndex**() {

Point2D point1 = **new** Point2D(**new** **double**[]{**0**, **0**});

Point2D point2 = **new** Point2D(**new** **double**[]{**1**, **0**});

Point2D point3 = **new** Point2D(**new** **double**[]{**0**, **1**});

NGon nGon = **new** NGon(**new** Point2D[]{point1, point2, point3});

assertThrows(IllegalArgumentException.class, () -> nGon.getP(-**1**),

"Метод getP должен выбрасывать исключение при отрицательном индексе");

}

**@Test**

**void** **testGetPWithIndexGreaterThanN**() {

Point2D point1 = **new** Point2D(**new** **double**[]{**0**, **0**});

Point2D point2 = **new** Point2D(**new** **double**[]{**1**, **0**});

Point2D point3 = **new** Point2D(**new** **double**[]{**0**, **1**});

NGon nGon = **new** NGon(**new** Point2D[]{point1, point2, point3});

assertThrows(IllegalArgumentException.class, () -> nGon.getP(**3**),

"Метод getP должен выбрасывать исключение при индексе больше или равном n");

}

**@Test**

**void** **testSetPWithNegativeIndex**() {

Point2D point1 = **new** Point2D(**new** **double**[]{**0**, **0**});

Point2D point2 = **new** Point2D(**new** **double**[]{**1**, **0**});

Point2D point3 = **new** Point2D(**new** **double**[]{**0**, **1**});

NGon nGon = **new** NGon(**new** Point2D[]{point1, point2, point3});

// Создаем новую точку

Point2D newPoint = **new** Point2D(**new** **double**[]{**2**, **2**});

assertThrows(IllegalArgumentException.class, () -> nGon.setP(newPoint, -**1**),

"Метод setP должен выбрасывать исключение при отрицательном индексе");

}

**@Test**

**void** **testSetPWithIndexGreaterThanN**() {

Point2D point1 = **new** Point2D(**new** **double**[]{**0**, **0**});

Point2D point2 = **new** Point2D(**new** **double**[]{**1**, **0**});

Point2D point3 = **new** Point2D(**new** **double**[]{**0**, **1**});

NGon nGon = **new** NGon(**new** Point2D[]{point1, point2, point3});

// Создаем новую точку

Point2D newPoint = **new** Point2D(**new** **double**[]{**2**, **2**});

assertThrows(IllegalArgumentException.class, () -> nGon.setP(newPoint, **3**),

"Метод setP должен выбрасывать исключение при индексе больше или равном n");

}

**@Test**

**void** **testCrossWithInvalidArgumentType**() {

Point2D point1 = **new** Point2D(**new** **double**[]{**0**, **0**});

Point2D point2 = **new** Point2D(**new** **double**[]{**1**, **0**});

Point2D point3 = **new** Point2D(**new** **double**[]{**0**, **1**});

NGon nGon = **new** NGon(**new** Point2D[]{point1, point2, point3});

// Создаем объект, который не является экземпляром NGon

IShape invalidShape = **new** IShape() {

**@Override**

**public** **double** **square**() {

**return** **0**;

}

**@Override**

**public** **double** **length**() {

**return** **0**;

}

**@Override**

**public** IShape **shift**(Point2D a) {

**return** **null**;

}

**@Override**

**public** IShape **rot**(**double** phi) {

**return** **null**;

}

**@Override**

**public** IShape **symAxis**(**int** i) {

**return** **null**;

}

**@Override**

**public** **boolean** **cross**(IShape i) {

**return** **false**;

}

};

// Проверяем, что метод cross выбрасывает исключение при недопустимом типе аргумента

assertThrows(IllegalArgumentException.class, () -> nGon.cross(invalidShape),

"Метод cross должен выбрасывать исключение при аргументе недопустимого типа");

}

**@Test**

**void** **testConstructorWithInvalidNumberOfVertices**() {

// Пытаемся создать NGon с недостаточным количеством вершин (меньше 3)

Point2D[] invalidVertices = **new** Point2D[]{**new** Point2D(), **new** Point2D()};

// Проверяем, что конструктор выбрасывает исключение при недостаточном количестве вершин

assertThrows(IllegalArgumentException.class, () -> **new** NGon(invalidVertices),

"Конструктор NGon должен выбрасывать исключение при недостаточном количестве вершин");

}

**@Test**

**void** **testSetP**() {

Point2D[] vertices = {

**new** **Point2D**(**new** **double**[]{**0.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, **4.0**}),

**new** **Point2D**(**new** **double**[]{**0.0**, **4.0**})

};

NGon ngon = **new** NGon(vertices);

Point2D newVertex = **new** Point2D(**new** **double**[]{**2.0**, **2.0**});

**int** index = **2**;

ngon.setP(newVertex, index);

assertEquals(newVertex, ngon.getP(index));

}

**@Test**

**void** **square**() {

Point2D[] vertices = {

**new** **Point2D**(**new** **double**[]{**0.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, **4.0**}),

**new** **Point2D**(**new** **double**[]{**0.0**, **4.0**})

};

NGon ngon = **new** NGon(vertices);

**double** expectedSquare = **16.0**; // Ожидаемая площадь

assertEquals(expectedSquare, ngon.square(), **0.001**);

}

**@Test**

**void** **length**() {

Point2D[] vertices = {

**new** **Point2D**(**new** **double**[]{**0.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, **4.0**}),

**new** **Point2D**(**new** **double**[]{**0.0**, **4.0**})

};

NGon ngon = **new** NGon(vertices);

**double** expectedLength = **16.0**; // Ожидаемая длина

assertEquals(expectedLength, ngon.length(), **0.001**);

}

**@Test**

**void** **shift**() {

Point2D[] vertices = {

**new** **Point2D**(**new** **double**[]{**0.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, **4.0**}),

**new** **Point2D**(**new** **double**[]{**0.0**, **4.0**})

};

NGon ngon = **new** NGon(vertices);

Point2D shiftVector = **new** Point2D(**new** **double**[]{**2.0**, **3.0**});

NGon shiftedNgon = (NGon) ngon.shift(shiftVector);

Point2D[] expectedVertices = {

**new** **Point2D**(**new** **double**[]{**2.0**, **3.0**}),

**new** **Point2D**(**new** **double**[]{**6.0**, **3.0**}),

**new** **Point2D**(**new** **double**[]{**6.0**, **7.0**}),

**new** **Point2D**(**new** **double**[]{**2.0**, **7.0**})

};

Point2D[] actualVertices = shiftedNgon.getP();

**for** (**int** i = **0**; i < vertices.length; i++) {

assertEquals(expectedVertices[i].getX(**0**), actualVertices[i].getX(**0**), **0.001**);

assertEquals(expectedVertices[i].getX(**1**), actualVertices[i].getX(**1**), **0.001**);

}

}

**@Test**

**void** **rot**() {

Point2D[] vertices = {

**new** **Point2D**(**new** **double**[]{**0.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, **4.0**}),

**new** **Point2D**(**new** **double**[]{**0.0**, **4.0**})

};

NGon ngon = **new** NGon(vertices);

**double** angle = Math.PI / **2**;

NGon rotatedNgon = (NGon) ngon.rot(angle);

Point2D[] expectedVertices = {

**new** **Point2D**(**new** **double**[]{**0.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**0.0**, **4.0**}),

**new** **Point2D**(**new** **double**[]{-**4.0**, **4.0**}),

**new** **Point2D**(**new** **double**[]{-**4.0**, **0.0**})

};

Point2D[] actualVertices = rotatedNgon.getP();

**for** (**int** i = **0**; i < vertices.length; i++) {

assertEquals(expectedVertices[i].getX(**0**), actualVertices[i].getX(**0**), **0.001**);

assertEquals(expectedVertices[i].getX(**1**), actualVertices[i].getX(**1**), **0.001**);

}

}

**@Test**

**void** **symAxis**() {

Point2D[] vertices = {

**new** **Point2D**(**new** **double**[]{**0.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, **4.0**}),

**new** **Point2D**(**new** **double**[]{**0.0**, **4.0**})

};

NGon ngon = **new** NGon(vertices);

**int** axis = **0**; // Ось x

NGon symmetrizedNgon = (NGon) ngon.symAxis(axis);

Point2D[] expectedVertices = {

**new** **Point2D**(**new** **double**[]{**0.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, -**4.0**}),

**new** **Point2D**(**new** **double**[]{**0.0**, -**4.0**})

};

Point2D[] actualVertices = symmetrizedNgon.getP();

**for** (**int** i = **0**; i < vertices.length; i++) {

assertEquals(expectedVertices[i].getX(**0**), actualVertices[i].getX(**0**), **0.001**);

assertEquals(expectedVertices[i].getX(**1**), actualVertices[i].getX(**1**), **0.001**);

}

}

**@Test**

**void** **cross**() {

Point2D[] vertices1 = {

**new** **Point2D**(**new** **double**[]{**0.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, **4.0**}),

**new** **Point2D**(**new** **double**[]{**0.0**, **4.0**})

};

NGon ngon1 = **new** NGon(vertices1);

Point2D[] vertices2 = {

**new** **Point2D**(**new** **double**[]{**2.0**, **2.0**}),

**new** **Point2D**(**new** **double**[]{**6.0**, **2.0**}),

**new** **Point2D**(**new** **double**[]{**6.0**, **6.0**}),

**new** **Point2D**(**new** **double**[]{**2.0**, **6.0**})

};

NGon ngon2 = **new** NGon(vertices2);

assertTrue(ngon1.cross(ngon2));

Point2D[] vertices3 = {

**new** **Point2D**(**new** **double**[]{**5.0**, **5.0**}),

**new** **Point2D**(**new** **double**[]{**9.0**, **5.0**}),

**new** **Point2D**(**new** **double**[]{**9.0**, **9.0**}),

**new** **Point2D**(**new** **double**[]{**5.0**, **9.0**})

};

NGon ngon3 = **new** NGon(vertices3);

assertFalse(ngon1.cross(ngon3));

}

**@Test**

**void** **testToString**() {

Point2D[] vertices = {

**new** **Point2D**(**new** **double**[]{**0.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, **4.0**}),

**new** **Point2D**(**new** **double**[]{**0.0**, **4.0**})

};

NGon ngon = **new** NGon(vertices);

String expectedString = "NGon(n=4, p=[Point(dim=2, x=[0.0, 0.0]), Point(dim=2, x=[4.0, 0.0]), Point(dim=2, x=[4.0, 4.0]), Point(dim=2, x=[0.0, 4.0])])";

assertEquals(expectedString, ngon.toString());

}

**@Test**

**public** **void** **testToDocument**() {

// Создаем массив точек для NGon

Point2D[] points = {

**new** **Point2D**(**new** **double**[]{**1.0**, **2.0**}),

**new** **Point2D**(**new** **double**[]{**3.0**, **4.0**}),

**new** **Point2D**(**new** **double**[]{**5.0**, **6.0**})

};

// Создаем экземпляр класса NGon

NGon nGon = **new** NGon(points);

Document document = nGon.toDocument();

// Проверяем, что документ создан корректно

assertEquals("NGon", document.getString("type"));

assertEquals(points.length, document.getInteger("n"));

// Проверяем, что массив точек вершин многоугольника корректно преобразуется в документ

Document[] pointDocuments = document.get("points", Document[].class);

assertNotNull(pointDocuments);

assertEquals(points.length, pointDocuments.length);

// Проверяем, что каждый объект Point2D в массиве точек корректно преобразуется в документ

**for** (**int** i = **0**; i < points.length; i++) {

Document pointDocument = pointDocuments[i];

assertNotNull(pointDocument);

assertEquals("Point2D", pointDocument.getString("type"));

assertEquals(**2**, pointDocument.getInteger("dim"));

assertNotNull(pointDocument.get("x")); // Проверяем, что поле x внутреннего Point2D не равно null

}

}

}

## **PolylineTest**

**package** org.figures.figures;

**import** **org.bson.Document**;

**import** **org.junit.jupiter.api.Test**;

**import** **org.figures.figures.points.Point2D**;

**import** **static** org.junit.jupiter.api.Assertions.\*;

**class** **PolylineTest** {

**@Test**

**void** **getN**() {

Point2D[] vertices = {

**new** **Point2D**(**new** **double**[]{**0.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, **4.0**}),

**new** **Point2D**(**new** **double**[]{**0.0**, **4.0**})

};

Polyline polyline = **new** Polyline(vertices);

**int** expectedN = **4**;

assertEquals(expectedN, polyline.getN());

}

**@Test**

**void** **testPolylineSquare**() {

Point2D[] points = {

**new** **Point2D**(**new** **double**[]{**0**, **0**}),

**new** **Point2D**(**new** **double**[]{**3**, **4**}),

**new** **Point2D**(**new** **double**[]{**6**, **0**})

};

Polyline polyline = **new** Polyline(points);

assertEquals(**0**, polyline.square(), **0.001**);

}

**@Test**

**void** **getP**() {

Point2D[] vertices = {

**new** **Point2D**(**new** **double**[]{**0.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, **4.0**}),

**new** **Point2D**(**new** **double**[]{**0.0**, **4.0**})

};

Polyline polyline = **new** Polyline(vertices);

Point2D[] expectedVertices = vertices;

assertArrayEquals(expectedVertices, polyline.getP());

}

**@Test**

**void** **testGetP**() {

Point2D[] vertices = {

**new** **Point2D**(**new** **double**[]{**0.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, **4.0**}),

**new** **Point2D**(**new** **double**[]{**0.0**, **4.0**})

};

Polyline polyline = **new** Polyline(vertices);

**int** index = **2**;

Point2D expectedPoint = vertices[index];

assertEquals(expectedPoint, polyline.getP(index));

}

**@Test**

**void** **setP**() {

Point2D[] vertices = {

**new** **Point2D**(**new** **double**[]{**0.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, **4.0**}),

**new** **Point2D**(**new** **double**[]{**0.0**, **4.0**})

};

Polyline polyline = **new** Polyline(vertices);

Point2D[] newVertices = {

**new** **Point2D**(**new** **double**[]{**1.0**, **1.0**}),

**new** **Point2D**(**new** **double**[]{**5.0**, **1.0**}),

**new** **Point2D**(**new** **double**[]{**5.0**, **5.0**}),

**new** **Point2D**(**new** **double**[]{**1.0**, **5.0**})

};

polyline.setP(newVertices);

assertArrayEquals(newVertices, polyline.getP());

}

**@Test**

**void** **testSetP**() {

Point2D[] vertices = {

**new** **Point2D**(**new** **double**[]{**0.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, **4.0**}),

**new** **Point2D**(**new** **double**[]{**0.0**, **4.0**})

};

Polyline polyline = **new** Polyline(vertices);

**int** index = **2**;

Point2D newPoint = **new** Point2D(**new** **double**[]{**2.0**, **2.0**});

polyline.setP(newPoint, index);

assertEquals(newPoint, polyline.getP(index));

}

**@Test**

**void** **length**() {

Point2D[] vertices = {

**new** **Point2D**(**new** **double**[]{**0.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, **4.0**}),

**new** **Point2D**(**new** **double**[]{**0.0**, **4.0**})

};

Polyline polyline = **new** Polyline(vertices);

**double** expectedLength = **12.0**;

assertEquals(expectedLength, polyline.length(), **0.001**);

}

**@Test**

**void** **shift**() {

Point2D[] vertices = {

**new** **Point2D**(**new** **double**[]{**0.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, **4.0**}),

**new** **Point2D**(**new** **double**[]{**0.0**, **4.0**})

};

Polyline polyline = **new** Polyline(vertices);

Point2D shiftVector = **new** Point2D(**new** **double**[]{**2.0**, **3.0**});

Polyline shiftedPolyline = (Polyline) polyline.shift(shiftVector);

Point2D[] expectedVertices = {

**new** **Point2D**(**new** **double**[]{**2.0**, **3.0**}),

**new** **Point2D**(**new** **double**[]{**6.0**, **3.0**}),

**new** **Point2D**(**new** **double**[]{**6.0**, **7.0**}),

**new** **Point2D**(**new** **double**[]{**2.0**, **7.0**})

};

Point2D[] actualVertices = shiftedPolyline.getP();

**for** (**int** i = **0**; i < vertices.length; i++) {

assertEquals(expectedVertices[i].getX(**0**), actualVertices[i].getX(**0**), **0.001**);

assertEquals(expectedVertices[i].getX(**1**), actualVertices[i].getX(**1**), **0.001**);

}

}

**@Test**

**void** **rot**() {

Point2D[] vertices = {

**new** **Point2D**(**new** **double**[]{**0.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, **4.0**}),

**new** **Point2D**(**new** **double**[]{**0.0**, **4.0**})

};

Polyline polyline = **new** Polyline(vertices);

**double** angle = Math.PI / **2**;

Polyline rotatedPolyline = (Polyline) polyline.rot(angle);

Point2D[] expectedVertices = {

**new** **Point2D**(**new** **double**[]{**0.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**0.0**, **4.0**}),

**new** **Point2D**(**new** **double**[]{-**4.0**, **4.0**}),

**new** **Point2D**(**new** **double**[]{-**4.0**, **0.0**})

};

Point2D[] actualVertices = rotatedPolyline.getP();

**for** (**int** i = **0**; i < expectedVertices.length; i++) {

**for** (**int** j = **0**; j < **2**; j++) {

assertEquals(expectedVertices[i].getX(j), actualVertices[i].getX(j), **0.001**);

}

}

}

**@Test**

**void** **symAxis**() {

Point2D[] vertices = {

**new** **Point2D**(**new** **double**[]{**0.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, **4.0**}),

**new** **Point2D**(**new** **double**[]{**0.0**, **4.0**})

};

Polyline polyline = **new** Polyline(vertices);

**int** axis = **0**;

Polyline symmetrizedPolyline = (Polyline) polyline.symAxis(axis);

Point2D[] expectedVertices = {

**new** **Point2D**(**new** **double**[]{**0.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, -**4.0**}),

**new** **Point2D**(**new** **double**[]{**0.0**, -**4.0**})

};

Point2D[] actualVertices = symmetrizedPolyline.getP();

**for** (**int** i = **0**; i < expectedVertices.length; i++) {

**for** (**int** j = **0**; j < **2**; j++) {

assertEquals(expectedVertices[i].getX(j), actualVertices[i].getX(j), **0.001**);

}

}

}

**@Test**

**void** **testPolylineCross**() {

Point2D[] points1 = {

**new** **Point2D**(**new** **double**[]{**1**, **1**}),

**new** **Point2D**(**new** **double**[]{**2**, **2**}),

**new** **Point2D**(**new** **double**[]{**3**, **3**})

};

Polyline polyline1 = **new** Polyline(points1);

Point2D[] points2 = {

**new** **Point2D**(**new** **double**[]{**2**, **1**}),

**new** **Point2D**(**new** **double**[]{**2**, **4**}),

**new** **Point2D**(**new** **double**[]{**4**, **4**})

};

Polyline polyline2 = **new** Polyline(points2);

assertThrows(IllegalArgumentException.class, () -> {

polyline1.cross(**new** Circle(**new** Point2D(), **1**));

}, "Метод cross должен выбрасывать IllegalArgumentException при передаче не Polyline объекта");

assertTrue(polyline1.cross(polyline2), "Метод cross должен возвращать true для пересекающихся Polyline");

Point2D[] points3 = {

**new** **Point2D**(**new** **double**[]{**5**, **5**}),

**new** **Point2D**(**new** **double**[]{**6**, **6**}),

**new** **Point2D**(**new** **double**[]{**7**, **7**})

};

Polyline polyline3 = **new** Polyline(points3);

assertFalse(polyline1.cross(polyline3), "Метод cross должен возвращать false для не пересекающихся Polyline");

}

**@Test**

**void** **testToString**() {

Point2D[] vertices = {

**new** **Point2D**(**new** **double**[]{**0.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, **4.0**}),

**new** **Point2D**(**new** **double**[]{**0.0**, **4.0**})

};

Polyline polyline = **new** Polyline(vertices);

String expectedString = "Polyline(n=4, p=[Point(dim=2, x=[0.0, 0.0]), Point(dim=2, x=[4.0, 0.0]), Point(dim=2, x=[4.0, 4.0]), Point(dim=2, x=[0.0, 4.0])])";

assertEquals(expectedString, polyline.toString());

}

**@Test**

**void** **testGetPInvalidIndex**() {

Point2D[] points = {

**new** **Point2D**(**new** **double**[]{**1**, **1**}),

**new** **Point2D**(**new** **double**[]{**2**, **2**}),

**new** **Point2D**(**new** **double**[]{**3**, **3**})

};

Polyline polyLine = **new** Polyline(points);

assertThrows(IllegalArgumentException.class, () -> {

polyLine.getP(-**1**);

}, "Метод getP должен выбрасывать IllegalArgumentException при отрицательном индексе");

assertThrows(IllegalArgumentException.class, () -> {

polyLine.getP(points.length);

}, "Метод getP должен выбрасывать IllegalArgumentException при индексе больше или равном n");

}

**@Test**

**void** **testSetPInvalidIndex**() {

Point2D[] points = {

**new** **Point2D**(**new** **double**[]{**1**, **1**}),

**new** **Point2D**(**new** **double**[]{**2**, **2**}),

**new** **Point2D**(**new** **double**[]{**3**, **3**})

};

Polyline polyLine = **new** Polyline(points);

assertThrows(IllegalArgumentException.class, () -> {

polyLine.setP(**new** Point2D(**new** **double**[]{**4**, **4**}), -**1**);

}, "Метод setP должен выбрасывать IllegalArgumentException при отрицательном индексе");

assertThrows(IllegalArgumentException.class, () -> {

polyLine.setP(**new** Point2D(**new** **double**[]{**4**, **4**}), points.length);

}, "Метод setP должен выбрасывать IllegalArgumentException при индексе больше или равном n");

}

**@Test**

**public** **void** **testToDocument**() {

// Создаем массив точек для NGon

Point2D[] points = {

**new** **Point2D**(**new** **double**[]{**1.0**, **2.0**}),

**new** **Point2D**(**new** **double**[]{**3.0**, **4.0**}),

**new** **Point2D**(**new** **double**[]{**5.0**, **6.0**})

};

// Создаем экземпляр класса NGon

Polyline polyline = **new** Polyline(points);

Document document = polyline.toDocument();

// Проверяем, что документ создан корректно

assertEquals("Polyline", document.getString("type"));

assertEquals(points.length, document.getInteger("n"));

// Проверяем, что массив точек вершин многоугольника корректно преобразуется в документ

Document[] pointDocuments = document.get("points", Document[].class);

assertNotNull(pointDocuments);

assertEquals(points.length, pointDocuments.length);

// Проверяем, что каждый объект Point2D в массиве точек корректно преобразуется в документ

**for** (**int** i = **0**; i < points.length; i++) {

Document pointDocument = pointDocuments[i];

assertNotNull(pointDocument);

assertEquals("Point2D", pointDocument.getString("type"));

assertEquals(**2**, pointDocument.getInteger("dim"));

assertNotNull(pointDocument.get("x")); // Проверяем, что поле x внутреннего Point2D не равно null

}

}

}

## **QGonTest**

**package** org.figures.figures;

**import** **org.bson.Document**;

**import** **org.junit.jupiter.api.BeforeAll**;

**import** **org.junit.jupiter.api.Test**;

**import** **org.figures.figures.points.Point2D**;

**import** **static** org.junit.jupiter.api.Assertions.assertEquals;

**import** **static** org.junit.jupiter.api.Assertions.assertNotNull;

**class** **QGonTest** {

**private** **static** QGon qGon;

**@BeforeAll**

**static** **void** **setUp**() {

Point2D[] vertices = {

**new** **Point2D**(**new** **double**[]{**0.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, **4.0**}),

**new** **Point2D**(**new** **double**[]{**0.0**, **4.0**})

};

qGon = **new** QGon(vertices);

}

**@Test**

**void** **square**() {

**double** expectedSquare = **16.0**;

assertEquals(expectedSquare, qGon.square(), **0.001**);

}

**@Test**

**void** **shift**() {

Point2D shiftVector = **new** Point2D(**new** **double**[]{**2.0**, **3.0**});

QGon shiftedQGon = (QGon) qGon.shift(shiftVector);

Point2D[] expectedVertices = {

**new** **Point2D**(**new** **double**[]{**2.0**, **3.0**}),

**new** **Point2D**(**new** **double**[]{**6.0**, **3.0**}),

**new** **Point2D**(**new** **double**[]{**6.0**, **7.0**}),

**new** **Point2D**(**new** **double**[]{**2.0**, **7.0**})

};

**for** (**int** i = **0**; i < expectedVertices.length; i++) {

**for** (**int** j = **0**; j < **2**; j++) {

assertEquals(expectedVertices[i].getX(j), shiftedQGon.getP()[i].getX(j), **0.001**);

}

}

}

**@Test**

**void** **rot**() {

**double** angle = Math.PI / **2**;

QGon rotatedQGon = (QGon) qGon.rot(angle);

Point2D[] expectedVertices = {

**new** **Point2D**(**new** **double**[]{**0.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**0.0**, **4.0**}),

**new** **Point2D**(**new** **double**[]{-**4.0**, **4.0**}),

**new** **Point2D**(**new** **double**[]{-**4.0**, **0.0**})

};

**for** (**int** i = **0**; i < expectedVertices.length; i++) {

**for** (**int** j = **0**; j < **2**; j++) {

assertEquals(expectedVertices[i].getX(j), rotatedQGon.getP()[i].getX(j), **0.001**);

}

}

}

**@Test**

**void** **symAxis**() {

**int** axis = **0**; // Ось x

QGon symmetrizedQGon = (QGon) qGon.symAxis(axis);

Point2D[] expectedVertices = {

**new** **Point2D**(**new** **double**[]{**0.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, -**4.0**}),

**new** **Point2D**(**new** **double**[]{**0.0**, -**4.0**})

};

**for** (**int** i = **0**; i < expectedVertices.length; i++) {

**for** (**int** j = **0**; j < **2**; j++) {

assertEquals(expectedVertices[i].getX(j), symmetrizedQGon.getP()[i].getX(j), **0.001**);

}

}

}

**@Test**

**public** **void** **testToDocument**() {

// Создаем массив точек для NGon

Point2D[] points = {

**new** **Point2D**(**new** **double**[]{**1.0**, **2.0**}),

**new** **Point2D**(**new** **double**[]{**3.0**, **4.0**}),

**new** **Point2D**(**new** **double**[]{**5.0**, **6.0**})

};

// Создаем экземпляр класса NGon

QGon qGon = **new** QGon(points);

Document document = qGon.toDocument();

// Проверяем, что документ создан корректно

assertEquals("QGon", document.getString("type"));

assertEquals(points.length, document.getInteger("n"));

// Проверяем, что массив точек вершин многоугольника корректно преобразуется в документ

Document[] pointDocuments = document.get("points", Document[].class);

assertNotNull(pointDocuments);

assertEquals(points.length, pointDocuments.length);

// Проверяем, что каждый объект Point2D в массиве точек корректно преобразуется в документ

**for** (**int** i = **0**; i < points.length; i++) {

Document pointDocument = pointDocuments[i];

assertNotNull(pointDocument);

assertEquals("Point2D", pointDocument.getString("type"));

assertEquals(**2**, pointDocument.getInteger("dim"));

assertNotNull(pointDocument.get("x")); // Проверяем, что поле x внутреннего Point2D не равно null

}

}

}

## **RectangleTest**

**package** org.figures.figures;

**import** **org.bson.Document**;

**import** **org.junit.jupiter.api.BeforeAll**;

**import** **org.junit.jupiter.api.Test**;

**import** **org.figures.figures.points.Point2D**;

**import** **static** org.junit.jupiter.api.Assertions.assertEquals;

**import** **static** org.junit.jupiter.api.Assertions.assertNotNull;

**class** **RectangleTest** {

**private** **static** Rectangle rectangle;

**@BeforeAll**

**static** **void** **setUp**() {

Point2D[] vertices = {

**new** **Point2D**(**new** **double**[]{**0.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, **2.0**}),

**new** **Point2D**(**new** **double**[]{**0.0**, **2.0**})

};

rectangle = **new** Rectangle(vertices);

}

**@Test**

**void** **square**() {

**double** expectedSquare = **8.0**;

assertEquals(expectedSquare, rectangle.square(), **0.001**);

}

**@Test**

**void** **shift**() {

Point2D shiftVector = **new** Point2D(**new** **double**[]{**2.0**, **3.0**});

Rectangle shiftedRectangle = (Rectangle) rectangle.shift(shiftVector);

Point2D[] expectedVertices = {

**new** **Point2D**(**new** **double**[]{**2.0**, **3.0**}),

**new** **Point2D**(**new** **double**[]{**6.0**, **3.0**}),

**new** **Point2D**(**new** **double**[]{**6.0**, **5.0**}),

**new** **Point2D**(**new** **double**[]{**2.0**, **5.0**})

};

**for** (**int** i = **0**; i < expectedVertices.length; i++) {

**for** (**int** j = **0**; j < **2**; j++) {

assertEquals(expectedVertices[i].getX(j), shiftedRectangle.getP()[i].getX(j), **0.001**);

}

}

}

**@Test**

**void** **rot**() {

**double** angle = Math.PI / **2**;

Rectangle rotatedRectangle = (Rectangle) rectangle.rot(angle);

Point2D[] expectedVertices = {

**new** **Point2D**(**new** **double**[]{**0.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**0.0**, **4.0**}),

**new** **Point2D**(**new** **double**[]{-**2.0**, **4.0**}),

**new** **Point2D**(**new** **double**[]{-**2.0**, **0.0**})

};

**for** (**int** i = **0**; i < expectedVertices.length; i++) {

**for** (**int** j = **0**; j < **2**; j++) {

assertEquals(expectedVertices[i].getX(j), rotatedRectangle.getP()[i].getX(j), **0.001**);

}

}

}

**@Test**

**void** **symAxis**() {

**int** axis = **0**;

Rectangle symmetrizedRectangle = (Rectangle) rectangle.symAxis(axis);

Point2D[] expectedVertices = {

**new** **Point2D**(**new** **double**[]{**0.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, -**2.0**}),

**new** **Point2D**(**new** **double**[]{**0.0**, -**2.0**})

};

**for** (**int** i = **0**; i < expectedVertices.length; i++) {

**for** (**int** j = **0**; j < **2**; j++) {

assertEquals(expectedVertices[i].getX(j), symmetrizedRectangle.getP()[i].getX(j), **0.001**);

}

}

}

**@Test**

**public** **void** **testToDocument**() {

// Создаем массив точек для NGon

Point2D[] points = {

**new** **Point2D**(**new** **double**[]{**1.0**, **2.0**}),

**new** **Point2D**(**new** **double**[]{**3.0**, **4.0**}),

**new** **Point2D**(**new** **double**[]{**5.0**, **6.0**})

};

// Создаем экземпляр класса NGon

Rectangle rectangle1 = **new** Rectangle(points);

Document document = rectangle1.toDocument();

// Проверяем, что документ создан корректно

assertEquals("Rectangle", document.getString("type"));

assertEquals(points.length, document.getInteger("n"));

// Проверяем, что массив точек вершин многоугольника корректно преобразуется в документ

Document[] pointDocuments = document.get("points", Document[].class);

assertNotNull(pointDocuments);

assertEquals(points.length, pointDocuments.length);

// Проверяем, что каждый объект Point2D в массиве точек корректно преобразуется в документ

**for** (**int** i = **0**; i < points.length; i++) {

Document pointDocument = pointDocuments[i];

assertNotNull(pointDocument);

assertEquals("Point2D", pointDocument.getString("type"));

assertEquals(**2**, pointDocument.getInteger("dim"));

assertNotNull(pointDocument.get("x")); // Проверяем, что поле x внутреннего Point2D не равно null

}

}

}

## **SegmentTest**

**package** org.figures.figures;

**import** **org.bson.Document**;

**import** **org.junit.jupiter.api.BeforeEach**;

**import** **org.junit.jupiter.api.Test**;

**import** **org.figures.figures.points.Point2D**;

**import** **org.junit.jupiter.params.ParameterizedTest**;

**import** **org.junit.jupiter.params.provider.CsvSource**;

**import** **static** org.junit.jupiter.api.Assertions.\*;

**class** **SegmentTest** {

**private** **static** Segment segment;

**@BeforeEach**

**void** **setUp**() {

Point2D start = **new** Point2D(**new** **double**[]{**0.0**, **0.0**});

Point2D finish = **new** Point2D(**new** **double**[]{**4.0**, **4.0**});

segment = **new** Segment(start, finish);

}

**@Test**

**void** **getStart**() {

Point2D expectedStart = **new** Point2D(**new** **double**[]{**0.0**, **0.0**});

Point2D actualStart = segment.getStart();

**for** (**int** i = **0**; i < expectedStart.getDim(); i++) {

assertEquals(expectedStart.getX(i), actualStart.getX(i), **0.001**);

}

}

**@Test**

**void** **testSegmentSquare**() {

Segment segment = **new** Segment(**new** Point2D(**new** **double**[]{**0**, **0**}), **new** Point2D(**new** **double**[]{**3**, **4**}));

assertEquals(**0**, segment.square(), **0.001**);

}

**@Test**

**void** **setStart**() {

Point2D newStart = **new** Point2D(**new** **double**[]{**1.0**, **1.0**});

segment.setStart(newStart);

assertEquals(newStart, segment.getStart());

}

**@Test**

**void** **getFinish**() {

Point2D expectedFinish = **new** Point2D(**new** **double**[]{**4.0**, **4.0**});

Point2D actualFinish = segment.getFinish();

**for** (**int** i = **0**; i < expectedFinish.getDim(); i++) {

assertEquals(expectedFinish.getX(i), actualFinish.getX(i), **0.001**);

}

}

**@Test**

**void** **setFinish**() {

Point2D newFinish = **new** Point2D(**new** **double**[]{**5.0**, **5.0**});

segment.setFinish(newFinish);

assertEquals(newFinish, segment.getFinish());

}

**@Test**

**void** **length**() {

**double** expectedLength = **5.65685**;

assertEquals(expectedLength, segment.length(), **0.001**);

}

**@Test**

**void** **shift**() {

Point2D shiftVector = **new** Point2D(**new** **double**[]{**2.0**, **3.0**});

Segment shiftedSegment = (Segment) segment.shift(shiftVector);

Point2D expectedStart = **new** Point2D(**new** **double**[]{**2.0**, **3.0**});

Point2D expectedFinish = **new** Point2D(**new** **double**[]{**6.0**, **7.0**});

Point2D actualStart = shiftedSegment.getStart();

Point2D actualFinish = shiftedSegment.getFinish();

**for** (**int** i = **0**; i < expectedStart.getDim(); i++) {

assertEquals(expectedStart.getX(i), actualStart.getX(i), **0.001**);

assertEquals(expectedFinish.getX(i), actualFinish.getX(i), **0.001**);

}

}

**@Test**

**void** **rot**() {

**double** angle = Math.PI / **2**;

Segment rotatedSegment = (Segment) segment.rot(angle);

Point2D expectedStart = **new** Point2D(**new** **double**[]{**0.0**, **0.0**});

Point2D expectedFinish = **new** Point2D(**new** **double**[]{-**4.0**, **4.0**});

Point2D actualStart = rotatedSegment.getStart();

Point2D actualFinish = rotatedSegment.getFinish();

**for** (**int** i = **0**; i < expectedStart.getDim(); i++) {

assertEquals(expectedStart.getX(i), actualStart.getX(i), **0.001**);

assertEquals(expectedFinish.getX(i), actualFinish.getX(i), **0.001**);

}

}

**@Test**

**void** **symAxis**() {

**int** axis = **0**;

Segment symmetrizedSegment = (Segment) segment.symAxis(axis);

Point2D expectedStart = **new** Point2D(**new** **double**[]{**0.0**, **0.0**});

Point2D expectedFinish = **new** Point2D(**new** **double**[]{**4.0**, -**4.0**});

Point2D actualStart = symmetrizedSegment.getStart();

Point2D actualFinish = symmetrizedSegment.getFinish();

**for** (**int** i = **0**; i < expectedStart.getDim(); i++) {

assertEquals(expectedStart.getX(i), actualStart.getX(i), **0.001**);

assertEquals(expectedFinish.getX(i), actualFinish.getX(i), **0.001**);

}

}

**@ParameterizedTest**

**@CsvSource**({

"1, 2, 3, 4, 5, 6, 7, 8, false", //ffff 0000

"1, 2, 3, 4, 5, 6, 7, 4, false", //ffft 0001

"1, 2, 4, 3, 5, 6, 4, 7, false", //fftf 0010

"1, 2, 3, 4, 5, 6, 3, 4, false", //fftt 0011

"1, 2, 3, 4, 5, 2, 7, 8, false", //0100

"1, 2, 3, 8, 5, 2, 7, 8, false", //0101

"1, 2, 8, 4, 5, 2, 8, 8, true", //0110

"1, 2, 3, 8, 5, 2, 3, 8, true", //0111

"1, 2, 3, 4, 1, 6, 7, 8, false", //1000

"1, 2, 3, 8, 1, 6, 7, 8, true", //1001

"1, 2, 6, 4, 1, 6, 6, 8, false", //1010

"1, 2, 9, 4, 1, 6, 9, 4, true", //1011

"1, 2, 3, 4, 1, 2, 7, 8, false", //1100

"1, 2, 3, 8, 1, 2, 7, 8, true", //1101

"1, 2, 7, 4, 1, 2, 7, 8, true", //1110

"1, 1, 3, 3, 1, 1, 3, 3, true", //1111

"0, 0, 3, 3, 1, 1, 4, 4, false", // Пересекаются внутри отрезков (ua=0.5, ub=0.5)

"0, 0, 3, 3, 4, 4, 5, 5, false", // Не пересекаются (ua=1.5, ub=-0.5)

"0, 0, 3, 3, -1, -1, 1, 1, false", // Пересекаются продолжения отрезков (ua=-0.5, ub=0.5)

"0, 0, 3, 3, -1, -1, -2, -2, false", // Не пересекаются продолжения отрезков (ua=-1.5, ub=-0.5)

"0, 0, 3, 3, 4, 0, 5, 1, false", // Не пересекаются (ua=1.25, ub=-0.25)

"0, 0, 3, 3, -1, 1, 1, -1, true", // Пересекаются продолжения отрезков (ua=-0.25, ub=1.25)

"0, 0, 3, 3, 3, 3, 5, 5, false", // Не пересекаются (ua=0, ub=2)

"0, 0, 3, 3, 3, 3, 2, 2, false", // Пересекаются внутри отрезков (ua=2, ub=0)

"0, 0, 3, 3, 1, 0, 3, 2, false", // Пересекаются внутри отрезков (ua=1, ub=0.5)

"0, 0, 3, 3, 2, 1, 4, 3, false", // Не пересекаются (ua=0.5, ub=1)

"0, 0, 3, 3, -1, 2, 1, 1, true", // Не пересекаются (ua=-0.25, ub=0.25)

"0, 0, 3, 3, -1, 3, 0, 2, false", // Пересекаются внутри отрезков (ua=-0.5, ub=0.75)

"0, 0, 3, 3, 3, 2, 5, 4, false", // Не пересекаются (ua=0.25, ub=2.25)

"0, 0, 3, 3, -1, 1, 2, -2, true", // Не пересекаются (ua=-0.75, ub=0)

"0, 0, 3, 3, 4, 2, 5, 3, false", // Пересекаются продолжения отрезков (ua=1.25, ub=0.75)

"0, 0, 3, 3, 4, 3, 5, 2, false", // Не пересекаются (ua=1.5, ub=-0.5)

"0, 0, 3, 3, 4, 0, 5, -1, false" // Оба значения ua и ub вне диапазона [0, 1]

})

**void** **testSegmentCross**(**double** x1, **double** y1, **double** x2, **double** y2, **double** x3, **double** y3, **double** x4, **double** y4, **boolean** expected) {

Point2D start1 = **new** Point2D(**new** **double**[]{x1, y1});

Point2D finish1 = **new** Point2D(**new** **double**[]{x2, y2});

Segment segment1 = **new** Segment(start1, finish1);

Point2D start2 = **new** Point2D(**new** **double**[]{x3, y3});

Point2D finish2 = **new** Point2D(**new** **double**[]{x4, y4});

Segment segment2 = **new** Segment(start2, finish2);

assertEquals(expected, segment1.cross(segment2));

assertThrows(IllegalArgumentException.class, () -> {

segment1.cross(**new** Circle(**new** Point2D(), **1**));

}, "Метод cross должен выбрасывать IllegalArgumentException при передаче не Segment объекта");

}

**@Test**

**void** **testToString**() {

String expectedString = "Segment(start=Point(dim=2, x=[0.0, 0.0]), finish=Point(dim=2, x=[4.0, 4.0]))";

String actualString = segment.toString();

assertEquals(expectedString, actualString);

}

**@Test**

**public** **void** **testToDocument**() {

// Создаем начальную и конечную точки для сегмента

Point2D start = **new** Point2D(**new** **double**[]{**1.0**, **2.0**});

Point2D finish = **new** Point2D(**new** **double**[]{**3.0**, **4.0**});

// Создаем экземпляр класса Segment

Segment segment = **new** Segment(start, finish);

Document document = segment.toDocument();

// Проверяем, что документ создан корректно

assertEquals("Segment", document.getString("type"));

// Проверяем, что начальная и конечная точки корректно преобразуются в документы

Document startDocument = document.get("start", Document.class);

Document finishDocument = document.get("finish", Document.class);

assertNotNull(startDocument);

assertNotNull(finishDocument);

assertEquals("Point2D", startDocument.getString("type"));

assertEquals("Point2D", finishDocument.getString("type"));

assertEquals(**2**, startDocument.getInteger("dim"));

assertEquals(**2**, finishDocument.getInteger("dim"));

assertNotNull(startDocument.get("x")); // Проверяем, что поле x внутреннего Point2D не равно null

assertNotNull(finishDocument.get("x"));

// Проверяем, что координаты начальной и конечной точек корректно преобразуются в документы

assertEquals(start.getX(), startDocument.get("x", **double**[].class));

assertEquals(finish.getX(), finishDocument.get("x", **double**[].class));

}

}

## **TGonTest**

**package** org.figures.figures;

**import** **org.bson.Document**;

**import** **org.junit.jupiter.api.Test**;

**import** **org.figures.figures.points.Point2D**;

**import** **static** org.junit.jupiter.api.Assertions.\*;

**import** **static** org.junit.jupiter.api.Assertions.assertNotNull;

**class** **TGonTest** {

**@Test**

**void** **square**() {

Point2D[] vertices = {

**new** **Point2D**(**new** **double**[]{**0.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**2.0**, **4.0**})

};

TGon tg = **new** TGon(vertices);

**double** expectedSquare = **8.0**;

assertEquals(expectedSquare, tg.square(), **0.001**);

}

**@Test**

**void** **testTGonConstructorWithInvalidCoordinates**() {

Point2D point1 = **new** Point2D(**new** **double**[]{**0**, **0**});

Point2D point2 = **new** Point2D(**new** **double**[]{**1**, **0**});

Point2D point3 = **new** Point2D(**new** **double**[]{**2**, **0**});

assertThrows(IllegalArgumentException.class, () -> **new** TGon(**new** Point2D[]{point1, point2, point3}),

"Конструктор TGon должен выбрасывать исключение при некорректных координатах");

}

**@Test**

**void** **shift**() {

Point2D[] vertices = {

**new** **Point2D**(**new** **double**[]{**0.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**2.0**, **4.0**})

};

TGon tg = **new** TGon(vertices);

Point2D shiftVector = **new** Point2D(**new** **double**[]{**2.0**, **3.0**});

TGon shiftedTgon = (TGon) tg.shift(shiftVector);

Point2D[] expectedVertices = {

**new** **Point2D**(**new** **double**[]{**2.0**, **3.0**}),

**new** **Point2D**(**new** **double**[]{**6.0**, **3.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, **7.0**})

};

**for** (**int** i = **0**; i < tg.getP().length; i++) {

Point2D expected = expectedVertices[i];

Point2D actual = shiftedTgon.getP()[i];

**for** (**int** j = **0**; j < expected.getDim(); j++) {

assertEquals(expected.getX(j), actual.getX(j), **0.001**);

}

}

}

**@Test**

**void** **rot**() {

Point2D[] vertices = {

**new** **Point2D**(**new** **double**[]{**0.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**2.0**, **4.0**})

};

TGon tg = **new** TGon(vertices);

**double** angle = Math.PI / **2**;

TGon rotatedTgon = (TGon) tg.rot(angle);

Point2D[] expectedVertices = {

**new** **Point2D**(**new** **double**[]{**0.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**0.0**, **4.0**}),

**new** **Point2D**(**new** **double**[]{-**4.0**, **2.0**})

};

**for** (**int** i = **0**; i < tg.getP().length; i++) {

Point2D expected = expectedVertices[i];

Point2D actual = rotatedTgon.getP()[i];

**for** (**int** j = **0**; j < expected.getDim(); j++) {

assertEquals(expected.getX(j), actual.getX(j), **0.001**);

}

}

}

**@Test**

**void** **symAxis**() {

//assertArrayEquals(expectedVertices, symmetrizedTgon.getP());

Point2D[] vertices = {

**new** **Point2D**(**new** **double**[]{**0.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**2.0**, **4.0**})

};

TGon tg = **new** TGon(vertices);

**int** axis = **0**;

TGon symmetrizedTgon = (TGon) tg.symAxis(axis);

Point2D[] expectedVertices = {

**new** **Point2D**(**new** **double**[]{**0.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**2.0**, -**4.0**})

};

**for** (**int** i = **0**; i < tg.getP().length; i++) {

Point2D expected = expectedVertices[i];

Point2D actual = symmetrizedTgon.getP()[i];

**for** (**int** j = **0**; j < expected.getDim(); j++) {

assertEquals(expected.getX(j), actual.getX(j), **0.001**);

}

}

}

**@Test**

**public** **void** **testToDocument**() {

// Создаем массив точек для NGon

Point2D[] points = {

**new** **Point2D**(**new** **double**[]{**0.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**3.0**, **4.0**}),

**new** **Point2D**(**new** **double**[]{**0.0**, **6.0**})

// Добавьте дополнительные точки при необходимости

};

// Создаем экземпляр класса NGon

TGon tGon = **new** TGon(points);

Document document = tGon.toDocument();

// Проверяем, что документ создан корректно

assertEquals("TGon", document.getString("type"));

assertEquals(points.length, document.getInteger("n"));

// Проверяем, что массив точек вершин многоугольника корректно преобразуется в документ

Document[] pointDocuments = document.get("points", Document[].class);

assertNotNull(pointDocuments);

assertEquals(points.length, pointDocuments.length);

// Проверяем, что каждый объект Point2D в массиве точек корректно преобразуется в документ

**for** (**int** i = **0**; i < points.length; i++) {

Document pointDocument = pointDocuments[i];

assertNotNull(pointDocument);

assertEquals("Point2D", pointDocument.getString("type"));

assertEquals(**2**, pointDocument.getInteger("dim"));

assertNotNull(pointDocument.get("x")); // Проверяем, что поле x внутреннего Point2D не равно null

}

}

}

## **TrapezeTest**

**package** org.figures.figures;

**import** **org.bson.Document**;

**import** **org.junit.jupiter.api.Test**;

**import** **org.figures.figures.points.Point2D**;

**import** **static** org.junit.jupiter.api.Assertions.assertEquals;

**import** **static** org.junit.jupiter.api.Assertions.assertNotNull;

**class** **TrapezeTest** {

**@Test**

**void** **square**() {

Point2D[] vertices = {

**new** **Point2D**(**new** **double**[]{**0.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**3.0**, **2.0**}),

**new** **Point2D**(**new** **double**[]{**1.0**, **2.0**})

};

Trapeze trapeze = **new** Trapeze(vertices);

**double** expectedSquare = **6.0**;

assertEquals(expectedSquare, trapeze.square(), **0.001**);

}

**@Test**

**void** **shift**() {

Point2D[] vertices = {

**new** **Point2D**(**new** **double**[]{**0.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**3.0**, **2.0**}),

**new** **Point2D**(**new** **double**[]{**1.0**, **2.0**})

};

Trapeze trapeze = **new** Trapeze(vertices);

Point2D shiftVector = **new** Point2D(**new** **double**[]{**2.0**, **3.0**});

Trapeze shiftedTrapeze = (Trapeze) trapeze.shift(shiftVector);

Point2D[] expectedVertices = {

**new** **Point2D**(**new** **double**[]{**2.0**, **3.0**}),

**new** **Point2D**(**new** **double**[]{**6.0**, **3.0**}),

**new** **Point2D**(**new** **double**[]{**5.0**, **5.0**}),

**new** **Point2D**(**new** **double**[]{**3.0**, **5.0**})

};

**for** (**int** i = **0**; i < trapeze.getP().length; i++) {

Point2D expected = expectedVertices[i];

Point2D actual = shiftedTrapeze.getP()[i];

**for** (**int** j = **0**; j < expected.getDim(); j++) {

assertEquals(expected.getX(j), actual.getX(j), **0.001**);

}

}

}

**@Test**

**void** **rot**() {

Point2D[] vertices = {

**new** **Point2D**(**new** **double**[]{**0.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**3.0**, **2.0**}),

**new** **Point2D**(**new** **double**[]{**1.0**, **2.0**})

};

Trapeze trapeze = **new** Trapeze(vertices);

**double** angle = Math.PI / **2**;

Trapeze rotatedTrapeze = (Trapeze) trapeze.rot(angle);

Point2D[] expectedVertices = {

**new** **Point2D**(**new** **double**[]{**0.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**0.0**, **4.0**}),

**new** **Point2D**(**new** **double**[]{-**2.0**, **3.0**}),

**new** **Point2D**(**new** **double**[]{-**2.0**, **1.0**})

};

**for** (**int** i = **0**; i < trapeze.getP().length; i++) {

Point2D expected = expectedVertices[i];

Point2D actual = rotatedTrapeze.getP()[i];

**for** (**int** j = **0**; j < expected.getDim(); j++) {

assertEquals(expected.getX(j), actual.getX(j), **0.001**);

}

}

}

**@Test**

**void** **symAxis**() {

Point2D[] vertices = {

**new** **Point2D**(**new** **double**[]{**0.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**3.0**, **2.0**}),

**new** **Point2D**(**new** **double**[]{**1.0**, **2.0**})

};

Trapeze trapeze = **new** Trapeze(vertices);

**int** axis = **0**;

Trapeze symmetrizedTrapeze = (Trapeze) trapeze.symAxis(axis);

Point2D[] expectedVertices = {

**new** **Point2D**(**new** **double**[]{**0.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**4.0**, **0.0**}),

**new** **Point2D**(**new** **double**[]{**3.0**, -**2.0**}),

**new** **Point2D**(**new** **double**[]{**1.0**, -**2.0**})

};

**for** (**int** i = **0**; i < trapeze.getP().length; i++) {

Point2D expected = expectedVertices[i];

Point2D actual = symmetrizedTrapeze.getP()[i];

**for** (**int** j = **0**; j < expected.getDim(); j++) {

assertEquals(expected.getX(j), actual.getX(j), **0.001**);

}

}

}

**@Test**

**public** **void** **testToDocument**() {

// Создаем массив точек для NGon

Point2D[] points = {

**new** **Point2D**(**new** **double**[]{**1.0**, **2.0**}),

**new** **Point2D**(**new** **double**[]{**3.0**, **4.0**}),

**new** **Point2D**(**new** **double**[]{**5.0**, **6.0**})

// Добавьте дополнительные точки при необходимости

};

// Создаем экземпляр класса NGon

Trapeze trapeze = **new** Trapeze(points);

Document document = trapeze.toDocument();

// Проверяем, что документ создан корректно

assertEquals("Trapeze", document.getString("type"));

assertEquals(points.length, document.getInteger("n"));

// Проверяем, что массив точек вершин многоугольника корректно преобразуется в документ

Document[] pointDocuments = document.get("points", Document[].class);

assertNotNull(pointDocuments);

assertEquals(points.length, pointDocuments.length);

// Проверяем, что каждый объект Point2D в массиве точек корректно преобразуется в документ

**for** (**int** i = **0**; i < points.length; i++) {

Document pointDocument = pointDocuments[i];

assertNotNull(pointDocument);

assertEquals("Point2D", pointDocument.getString("type"));

assertEquals(**2**, pointDocument.getInteger("dim"));

assertNotNull(pointDocument.get("x")); // Проверяем, что поле x внутреннего Point2D не равно null

}

}

}

## **IShape**

**package** org.figures.interfaces;

**import** **org.figures.figures.points.Point2D**;

**public** **interface** **IShape** {

**double** **square**(); // площадь фигуры

**double** **length**(); // длина/периметр фигуры

IShape **shift**(Point2D a); // сдвиг фигуры на вектор, заданный объектом класса Classes.Point2D

IShape **rot**(**double** phi); // поворот фигуры на угол phi в радианах относительно начала координат

IShape **symAxis**(**int** i); // симметрия фигуры относительно оси под заданным номером

**boolean** **cross**(IShape i); // проверка на пересечение с другой фигурой

}

## **OpenFigure**

**package** org.figures.abstractclasses;

**import** **org.figures.figures.points.Point2D**;

**import** **org.figures.interfaces.IShape**;

**public** **abstract** **class** **OpenFigure** **implements** IShape {

**@Override**

**public** **double** **square**() {

**return** **0**;

}

**public** **abstract** **double** **length**();

**public** **abstract** IShape **shift**(Point2D a);

**public** **abstract** IShape **rot**(**double** phi);

**public** **abstract** IShape **symAxis**(**int** i);

**public** **abstract** **boolean** **cross**(IShape i);

}

## **OpenFigureTest**

**package** org.figures.abstractclasses;

**import** **org.figures.figures.points.Point2D**;

**import** **org.figures.interfaces.IShape**;

**import** **org.junit.jupiter.api.Test**;

**import** **static** org.junit.jupiter.api.Assertions.\*;

**class** **OpenFigureTest** {

**@Test**

**void** **testSquare**() {

OpenFigure openFigure = **new** OpenFigure() {

**@Override**

**public** **double** **length**() {

**return** **0**;

}

**@Override**

**public** IShape **shift**(Point2D a) {

**return** **null**;

}

**@Override**

**public** IShape **rot**(**double** phi) {

**return** **null**;

}

**@Override**

**public** IShape **symAxis**(**int** i) {

**return** **null**;

}

**@Override**

**public** **boolean** **cross**(IShape i) {

**return** **false**;

}

};

**double** result = openFigure.square();

assertEquals(**0**, result, **0.001**);

}

}

## **NullPointException**

**package** com.figures.exceptions;

**public** **class** **NullInputException** **extends** Exception {

**public** **NullInputException**() { }

**@Override**

**public** String **toString**() {

**return** "Выбраны не все значения!";

}

}

## **AddFiguresController**

**package** com.figures.figuresfx;

**import** **com.figures.exceptions.NullInputException**;

**import** **com.figures.storage.Storage**;

**import** **javafx.beans.value.ChangeListener**;

**import** **javafx.beans.value.ObservableValue**;

**import** **javafx.event.ActionEvent**;

**import** **javafx.fxml.FXML**;

**import** **javafx.scene.Node**;

**import** **javafx.scene.canvas.Canvas**;

**import** **javafx.scene.canvas.GraphicsContext**;

**import** **javafx.scene.control.\***;

**import** **javafx.scene.layout.Pane**;

**import** **javafx.scene.paint.Color**;

**import** **javafx.stage.Stage**;

**import** **lombok.extern.java.Log**;

**import** **org.figures.figures.\***;

**import** **org.figures.figures.points.Point2D**;

**import** **java.util.Arrays**;

**import** **java.util.List**;

**@Log**

**public** **class** **AddFiguresController** {

**private** MainController mainController;

**private** CrossFiguresController crossFiguresController;

**private** MoveFiguresController moveFiguresController;

**private** DeleteFiguresController deleteFiguresController;

**public** **void** **setControllers**(MainController mainController, CrossFiguresController crossFiguresController, MoveFiguresController moveFiguresController, DeleteFiguresController deleteFiguresController) {

**this**.mainController = mainController;

**this**.crossFiguresController = crossFiguresController;

**this**.moveFiguresController = moveFiguresController;

**this**.deleteFiguresController = deleteFiguresController;

}

**private** FiguresApp application;

**public** **void** **setApplication**(FiguresApp application) {

**this**.application = application;

}

**public** **AddFiguresController**(){}

**@FXML**

**private** Spinner<Integer> numericUpDown1;

**@FXML**

**private** ComboBox<String> comboBox1;

**@FXML**

**private** Pane pn1;

**@FXML**

**private** Pane pn2;

**@FXML**

**private** Pane pn3;

**@FXML**

**private** Pane pn4;

**@FXML**

**private** Pane pn5;

**@FXML**

**private** Pane pn6;

**@FXML**

**private** Pane pn7;

**@FXML**

**private** Pane pn8;

**@FXML**

**private** Pane pn9;

**@FXML**

**private** Pane pn10;

**@FXML**

**private** Pane pn11;

**@FXML**

**private** Pane pn12;

**@FXML**

**private** Pane pn13;

**@FXML**

**private** Pane pn14;

**@FXML**

**private** Pane pn15;

**@FXML**

**private** Pane pn16;

**@FXML**

**private** Pane pn17;

**@FXML**

**private** Pane pn18;

**@FXML**

**private** TextField textBox1;

**@FXML**

**private** TextField textBox3;

**@FXML**

**private** TextField textBox2;

**@FXML**

**private** TextField textBox4;

**@FXML**

**private** TextField textBox5;

**@FXML**

**private** TextField textBox6;

**@FXML**

**private** TextField textBox7;

**@FXML**

**private** TextField textBox8;

**@FXML**

**private** TextField textBox9;

**@FXML**

**private** TextField textBox10;

**@FXML**

**private** TextField textBox11;

**@FXML**

**private** TextField textBox12;

**@FXML**

**private** TextField textBox13;

**@FXML**

**private** TextField textBox14;

**@FXML**

**private** TextField textBox15;

**@FXML**

**private** TextField textBox16;

**@FXML**

**private** TextField textBox17;

**@FXML**

**private** TextField textBox18;

**@FXML**

**private** TextField textBox19;

**@FXML**

**private** TextField textBox20;

**@FXML**

**private** TextField textBox21;

**@FXML**

**private** TextField textBox22;

**@FXML**

**private** TextField textBox23;

**@FXML**

**private** TextField textBox24;

**@FXML**

**private** TextField textBox25;

**@FXML**

**private** TextField textBox26;

**@FXML**

**private** TextField textBox27;

**@FXML**

**private** TextField textBox28;

**@FXML**

**private** TextField textBox29;

**@FXML**

**private** TextField textBox30;

**@FXML**

**private** TextField textBox31;

**@FXML**

**private** TextField textBox32;

**@FXML**

**private** TextField textBox33;

**@FXML**

**private** TextField textBox34;

**@FXML**

**private** TextField textBox35;

**@FXML**

**private** TextField textBox36;

**@FXML**

**private** TextField textBox37;

**@FXML**

**private** Label label1;

**public** **void** **initialize**(){

// Создание SpinnerValueFactory и настройка минимального и максимального значения

SpinnerValueFactory<Integer> valueFactory = **new** SpinnerValueFactory.IntegerSpinnerValueFactory(**1**, **100**, **1**);

// Установка SpinnerValueFactory для Spinner

numericUpDown1.setValueFactory(valueFactory);

comboBox1.getItems().addAll("Отрезок",

"Ломанная",

"Окружность",

"Многоугольник",

"Треугольник",

"Четырёхугольник",

"Прямоугольник",

"Трапеция");

SpinnerValueFactory<Integer> valueFactory\_nup = **new** SpinnerValueFactory.IntegerSpinnerValueFactory(

**1**, // Минимальное значение

**18**, // Максимальное значение

**1** // Начальное значение

);

numericUpDown1.setValueFactory(valueFactory\_nup);

// Устанавливаем слушатель событий на изменение значения Spinner

numericUpDown1.valueProperty().addListener(**new** ChangeListener<Integer>() {

**@Override**

**public** **void** **changed**(ObservableValue<? **extends** Integer> observable, Integer oldValue, Integer newValue) {

// В этом методе обрабатываем изменение значения Spinner

**int** value = numericUpDown1.getValue();

List<Pane> panes = Arrays.asList(pn1, pn2, pn3, pn4, pn5, pn6, pn7, pn8, pn9, pn10, pn11, pn12, pn13, pn14, pn15, pn16, pn17, pn18);

**for** (**int** i = **0**; i < **18**; i++) {

panes.get(i).setVisible(i < value);

}

Storage.pointsCount = value;

}

});

}

**@FXML**

**public** **void** **comboBox1\_SelectedIndexChanged**(ActionEvent actionEvent) {

numericUpDown1.setVisible(**true**);

pn1.setVisible(**true**);

String selectedValue = comboBox1.getValue();

label1.setVisible(**false**);

textBox1.setVisible(**false**);

numericUpDown1.setDisable(**false**);

**switch** (selectedValue) {

**case** "Отрезок":

numericUpDown1.getValueFactory().setValue(**2**);

numericUpDown1.setDisable(**true**);

**break**;

**case** "Ломанная":

**break**;

**case** "Треугольник":

numericUpDown1.getValueFactory().setValue(**3**);

numericUpDown1.setDisable(**true**);

**break**;

**case** "Окружность":

label1.setVisible(**true**);

textBox1.setVisible(**true**);

numericUpDown1.getValueFactory().setValue(**1**);

numericUpDown1.setDisable(**true**);

**break**;

**case** "Многоугольник":

**break**;

**case** "Трапеция", "Четырёхугольник", "Прямоугольник":

numericUpDown1.getValueFactory().setValue(**4**);

label1.setVisible(**false**);

textBox1.setVisible(**false**);

numericUpDown1.setDisable(**true**);

**break**;

}

}

// Добавить

**@FXML**

**private** **void** **button4\_Click**(ActionEvent event) {

**try** {

**if** (comboBox1.getSelectionModel().getSelectedIndex() == -**1**) {

**throw** **new** **NullInputException**();

}

update(); // Обновляет кол-во вершин

Storage.getListPoints().clear();

**switch** (Storage.getPointsCount()) {

**case** **1**:

Storage.getListPoints().add(Double.parseDouble(textBox3.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox2.getText()));

**break**;

**case** **2**:

Storage.getListPoints().add(Double.parseDouble(textBox3.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox2.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox8.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox9.getText()));

**break**;

**case** **3**:

Storage.getListPoints().add(Double.parseDouble(textBox3.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox2.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox8.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox9.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox10.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox11.getText()));

**break**;

**case** **4**:

Storage.getListPoints().add(Double.parseDouble(textBox3.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox2.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox8.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox9.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox10.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox11.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox12.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox13.getText()));

**break**;

**case** **5**:

Storage.getListPoints().add(Double.parseDouble(textBox3.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox2.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox8.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox9.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox10.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox11.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox12.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox13.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox14.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox15.getText()));

**break**;

**case** **6**:

Storage.getListPoints().add(Double.parseDouble(textBox3.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox2.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox8.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox9.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox10.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox11.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox12.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox13.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox14.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox15.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox16.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox17.getText()));

**break**;

**case** **7**:

Storage.getListPoints().add(Double.parseDouble(textBox3.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox2.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox8.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox9.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox10.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox11.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox12.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox13.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox14.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox15.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox16.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox17.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox4.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox5.getText()));

**break**;

**case** **8**:

Storage.getListPoints().add(Double.parseDouble(textBox3.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox2.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox8.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox9.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox10.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox11.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox12.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox13.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox14.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox15.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox16.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox17.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox4.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox5.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox18.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox19.getText()));

**break**;

**case** **9**:

Storage.getListPoints().add(Double.parseDouble(textBox3.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox2.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox8.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox9.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox10.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox11.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox12.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox13.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox14.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox15.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox16.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox17.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox4.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox5.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox18.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox19.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox20.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox21.getText()));

**break**;

**case** **10**:

Storage.getListPoints().add(Double.parseDouble(textBox3.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox2.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox8.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox9.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox10.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox11.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox12.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox13.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox14.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox15.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox16.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox17.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox4.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox5.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox18.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox19.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox20.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox21.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox22.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox23.getText()));

**break**;

**case** **11**:

Storage.getListPoints().add(Double.parseDouble(textBox3.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox2.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox8.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox9.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox10.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox11.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox12.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox13.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox14.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox15.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox16.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox17.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox4.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox5.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox18.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox19.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox20.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox21.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox24.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox25.getText()));

**break**;

**case** **12**:

Storage.getListPoints().add(Double.parseDouble(textBox3.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox2.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox8.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox9.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox10.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox11.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox12.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox13.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox14.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox15.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox16.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox17.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox4.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox5.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox18.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox19.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox20.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox21.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox24.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox25.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox26.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox27.getText()));

**break**;

**case** **13**:

Storage.getListPoints().add(Double.parseDouble(textBox3.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox2.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox8.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox9.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox10.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox11.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox12.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox13.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox14.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox15.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox16.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox17.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox4.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox5.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox18.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox19.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox20.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox21.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox24.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox25.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox26.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox27.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox6.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox7.getText()));

**break**;

**case** **14**:

Storage.getListPoints().add(Double.parseDouble(textBox3.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox2.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox8.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox9.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox10.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox11.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox12.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox13.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox14.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox15.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox16.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox17.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox4.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox5.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox18.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox19.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox20.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox21.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox24.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox25.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox26.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox27.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox6.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox7.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox28.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox29.getText()));

**break**;

**case** **15**:

Storage.getListPoints().add(Double.parseDouble(textBox3.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox2.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox8.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox9.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox10.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox11.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox12.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox13.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox14.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox15.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox16.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox17.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox4.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox5.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox18.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox19.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox20.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox21.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox24.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox25.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox26.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox27.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox6.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox7.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox28.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox29.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox30.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox31.getText()));

**break**;

**case** **16**:

Storage.getListPoints().add(Double.parseDouble(textBox3.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox2.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox8.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox9.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox10.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox11.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox12.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox13.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox14.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox15.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox16.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox17.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox4.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox5.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox18.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox19.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox20.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox21.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox24.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox25.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox26.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox27.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox6.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox7.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox28.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox29.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox30.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox31.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox32.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox33.getText()));

**break**;

**case** **17**:

Storage.getListPoints().add(Double.parseDouble(textBox3.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox2.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox8.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox9.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox10.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox11.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox12.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox13.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox14.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox15.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox16.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox17.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox4.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox5.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox18.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox19.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox20.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox21.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox24.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox25.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox26.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox27.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox6.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox7.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox28.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox29.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox30.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox31.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox32.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox33.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox34.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox35.getText()));

**case** **18**:

Storage.getListPoints().add(Double.parseDouble(textBox3.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox2.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox8.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox9.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox10.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox11.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox12.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox13.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox14.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox15.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox16.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox17.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox4.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox5.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox18.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox19.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox20.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox21.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox24.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox25.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox26.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox27.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox6.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox7.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox28.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox29.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox30.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox31.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox32.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox33.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox34.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox35.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox36.getText()));

Storage.getListPoints().add(Double.parseDouble(textBox37.getText()));

**break**;

}

String figure = comboBox1.getValue();

**switch** (figure) {

**case** "Окружность":

Circle circle = **new** Circle(**new** Point2D(**new** **double**[]{Double.parseDouble(Storage.getListPoints().get(**0**).toString()), Double.parseDouble(Storage.getListPoints().get(**1**).toString())}), Double.parseDouble(textBox1.getText()));

Storage.getListFigures().add(circle);

Storage.nameFigures.add("Окружность");

Storage.figuresCount++;

log.info("Добавлена окружность " + circle);

**break**;

**case** "Отрезок":

Segment segment = **new** Segment(**new** Point2D(**new** **double**[]{Double.parseDouble(Storage.getListPoints().get(**0**).toString()), Double.parseDouble(Storage.getListPoints().get(**1**).toString())}),

**new** **Point2D**(**new** **double**[]{Double.parseDouble(Storage.getListPoints().get(**2**).toString()),

Double.parseDouble(Storage.getListPoints().get(**3**).toString())}));

Storage.getListFigures().add(segment);

Storage.nameFigures.add("Отрезок");

Storage.figuresCount++;

log.info("Добавлен отрезок " + segment);

**break**;

**case** "Ломанная": // Если фигура - полилиния

**case** "Многоугольник": // Если фигура - правильный n-угольник

**case** "Четырёхугольник": // Если фигура - квадрат

**case** "Треугольник": // Если фигура - равнобедренный треугольник

**case** "Трапеция": // Если фигура - трапеция

**case** "Прямоугольник": // Если фигура - прямоугольник

**int** k = Integer.parseInt(numericUpDown1.getValue().toString());

Point2D[] coords = **new** Point2D[k];

**for** (**int** i = **0**, j = **0**; i < k; i++, j += **2**) {

coords[i] = **new** Point2D(**new** **double**[]{Storage.getListPoints().get(j), Storage.getListPoints().get(j + **1**)});

}

// Добавление фигуры в список в зависимости от типа

**if** (figure == "Ломанная") {

Polyline polyline = **new** Polyline(coords);

Storage.getListFigures().add(polyline);

Storage.nameFigures.add("Ломанная");

Storage.figuresCount++;

log.info("Добавлена ломанная " + polyline);

}

**if** (figure == "Многоугольник") {

NGon nGon = **new** NGon(coords);

Storage.getListFigures().add(nGon);

Storage.nameFigures.add("Многоугольник");

Storage.figuresCount++;

log.info("Добавлен многоугольник " + nGon);

}

**if** (figure == "Четырёхугольник") {

QGon qGon = **new** QGon(coords);

Storage.getListFigures().add(qGon);

Storage.nameFigures.add("Четырёхугольник");

Storage.figuresCount++;

log.info("Добавлен четырёхугольник " + qGon);

}

**if** (figure == "Треугольник") {

TGon tGon = **new** TGon(coords);

Storage.getListFigures().add(tGon);

Storage.nameFigures.add("Треугольник");

Storage.figuresCount++;

log.info("Добавлен треугольник " + tGon);

}

**if** (figure == "Трапеция") {

Trapeze trapeze = **new** Trapeze(coords);

Storage.getListFigures().add(trapeze);

Storage.nameFigures.add("Трапеция");

Storage.figuresCount++;

log.info("Добавлена трапеция " + trapeze);

}

**if** (figure == "Прямоугольник") {

Rectangle rectangle = **new** Rectangle(coords);

Storage.getListFigures().add(rectangle);

Storage.nameFigures.add("Прямоугольник");

Storage.figuresCount++;

log.info("Добавлен прямоугольник " + rectangle);

}

**break**;

}

Canvas canvas = mainController.canvas;

GraphicsContext gc = mainController.canvas.getGraphicsContext2D();

gc.clearRect(**0**, **0**, canvas.getWidth(), canvas.getHeight());

application.drawAxes(canvas.getWidth(), canvas.getHeight());

**for**(**int** i = **0**; i < Storage.figuresCount; i++){

mainController.draw(Storage.getListFigures().get(i), Color.BLACK);

}

Alert alert = **new** Alert(Alert.AlertType.INFORMATION);

alert.setTitle("Успех");

alert.setHeaderText(**null**);

alert.setContentText("Добавление завершено");

alert.showAndWait();

} **catch** (NullInputException ex) {

Alert alert = **new** Alert(Alert.AlertType.WARNING);

alert.setTitle("Ошибка");

alert.setHeaderText(**null**);

alert.setContentText(ex.toString());

alert.showAndWait();

} **catch** (NumberFormatException ex) {

Alert alert = **new** Alert(Alert.AlertType.WARNING);

alert.setTitle("Ошибка");

alert.setHeaderText(**null**);

alert.setContentText("Выбраны не все значения!");

alert.showAndWait();

} **catch** (IllegalArgumentException ex) {

Alert alert = **new** Alert(Alert.AlertType.WARNING);

alert.setTitle("Ошибка");

alert.setHeaderText(**null**);

alert.setContentText(ex.getMessage());

alert.showAndWait();

}

}

**public** **void** **update**() {

Storage.setPointsCount(numericUpDown1.getValue());

}

**public** **void** **buttonCancel\_Click**(ActionEvent actionEvent) {

Stage stage = (Stage) ((Node) actionEvent.getSource()).getScene().getWindow();

// Закрыть текущую сцену (форму)

stage.close();

}

}

## **CrossFiguresController**

**package** com.figures.figuresfx;

**import** **com.figures.exceptions.EqualsException**;

**import** **com.figures.exceptions.NullInputException**;

**import** **com.figures.storage.Storage**;

**import** **javafx.collections.FXCollections**;

**import** **javafx.collections.ObservableList**;

**import** **javafx.event.ActionEvent**;

**import** **javafx.fxml.FXML**;

**import** **javafx.scene.Node**;

**import** **javafx.scene.canvas.GraphicsContext**;

**import** **javafx.scene.control.Alert**;

**import** **javafx.scene.control.ComboBox**;

**import** **javafx.scene.paint.Color**;

**import** **javafx.stage.Stage**;

**import** **lombok.extern.java.Log**;

**import** **org.figures.interfaces.IShape**;

**import** **java.util.ArrayList**;

**import** **java.util.List**;

**import** **static** com.figures.storage.Storage.nameFigures;

**@Log**

**public** **class** **CrossFiguresController** {

**public** **static** List<IShape> listFigures = **new** ArrayList<>();

**public** **static** List<String> names = **new** ArrayList<>();

**@FXML**

**public** ComboBox<String> comboBox5;

**@FXML**

**public** ComboBox<String> comboBox6;

**@FXML**

**public** ComboBox<String> comboBox7;

**private** MainController mainController;

**private** AddFiguresController addFiguresController;

**private** MoveFiguresController moveFiguresController;

**private** DeleteFiguresController deleteFiguresController;

**private** FiguresApp application;

**public** **void** **setApplication**(FiguresApp application) {

**this**.application = application;

}

**public** **void** **setControllers**(MainController mainController, AddFiguresController addFiguresController, MoveFiguresController moveFiguresController, DeleteFiguresController deleteFiguresController) {

**this**.mainController = mainController;

**this**.addFiguresController = addFiguresController;

**this**.moveFiguresController = moveFiguresController;

**this**.deleteFiguresController = deleteFiguresController;

}

**public** **void** **initialize**(){

comboBox7.getItems().addAll("Отрезок",

"Ломанная",

"Окружность",

"Многоугольник",

"Треугольник",

"Четырёхугольник",

"Прямоугольник",

"Трапеция");

}

**@FXML**

**public** **void** **comboBox7\_move**(ActionEvent actionEvent) {

listFigures.clear();

names.clear();

// создание наблюдаемого списка строк

ObservableList<String> items = FXCollections.observableArrayList();

String value = comboBox7.getValue();

**switch** (value){

**case** "Отрезок":

items.clear();

**for**(**int** i = **0**; i < Storage.figuresCount; i++){

**if**(Storage.nameFigures.get(i).equals("Отрезок")){

items.add("Отрезок " + String.valueOf(Storage.listFigures.get(i)));

listFigures.add(Storage.listFigures.get(i));

}

}

comboBox5.setItems(items);

comboBox6.setItems(items);

**break**;

**case** "Ломанная":

items.clear();

**for**(**int** i = **0**; i < Storage.figuresCount; i++){

**if**(Storage.nameFigures.get(i).equals("Ломанная")){

items.add("Ломанная " + String.valueOf(Storage.listFigures.get(i)));

listFigures.add(Storage.listFigures.get(i));

}

}

comboBox5.setItems(items);

comboBox6.setItems(items);

**break**;

**case** "Окружность":

items.clear();

**for**(**int** i = **0**; i < Storage.figuresCount; i++){

**if**(Storage.nameFigures.get(i).equals("Окружность")){

items.add("Окружность " + String.valueOf(Storage.listFigures.get(i)));

listFigures.add(Storage.listFigures.get(i));

}

}

comboBox5.setItems(items);

comboBox6.setItems(items);

**break**;

**case** "Многоугольник":

items.clear();

**for**(**int** i = **0**; i < Storage.figuresCount; i++){

**if**(Storage.nameFigures.get(i).equals("Многоугольник")){

items.add("Многоугольник " + String.valueOf(Storage.listFigures.get(i)));

listFigures.add(Storage.listFigures.get(i));

}

}

comboBox5.setItems(items);

comboBox6.setItems(items);

**break**;

**case** "Треугольник":

items.clear();

**for**(**int** i = **0**; i < Storage.figuresCount; i++){

**if**(Storage.nameFigures.get(i).equals("Треугольник")){

items.add("Треугольник " + String.valueOf(Storage.listFigures.get(i)));

listFigures.add(Storage.listFigures.get(i));

}

}

comboBox5.setItems(items);

comboBox6.setItems(items);

**break**;

**case** "Четырёхугольник":

items.clear();

**for**(**int** i = **0**; i < Storage.figuresCount; i++){

**if**(Storage.nameFigures.get(i).equals("Четырёхугольник")){

items.add("Четырёхугольник " + String.valueOf(Storage.listFigures.get(i)));

listFigures.add(Storage.listFigures.get(i));

}

}

comboBox5.setItems(items);

comboBox6.setItems(items);

**break**;

**case** "Прямоугольник":

items.clear();

**for**(**int** i = **0**; i < Storage.figuresCount; i++){

**if**(Storage.nameFigures.get(i).equals("Прямоугольник")){

items.add("Прямоугольник " + String.valueOf(Storage.listFigures.get(i)));

listFigures.add(Storage.listFigures.get(i));

}

}

comboBox5.setItems(items);

comboBox6.setItems(items);

**break**;

**case** "Трапеция":

items.clear();

**for**(**int** i = **0**; i < Storage.figuresCount; i++){

**if**(Storage.nameFigures.get(i).equals("Трапеция")){

items.add("Трапеция " + String.valueOf(Storage.listFigures.get(i)));

listFigures.add(Storage.listFigures.get(i));

}

}

comboBox5.setItems(items);

comboBox6.setItems(items);

**break**;

}

}

// Пересечение

**@FXML**

**private** **void** **button7\_Click**(ActionEvent actionEvent){

**try** {

**if** (comboBox5.getSelectionModel().getSelectedIndex() == -**1** || comboBox6.getSelectionModel().getSelectedIndex() == -**1**) {

**throw** **new** **NullInputException**();

}

**if** (comboBox5.getSelectionModel().getSelectedIndex() == comboBox6.getSelectionModel().getSelectedIndex()) {

**throw** **new** **EqualsException**();

}

**if**(moveFiguresController.comboBox3.getSelectionModel().getSelectedIndex() < **0**){

moveFiguresController.comboBox3.getSelectionModel().select(-**1**);

}

**if** (deleteFiguresController.comboBox4.getSelectionModel().getSelectedIndex() < **0**) {

deleteFiguresController.comboBox4.getSelectionModel().select(**0**);

}

**if** (comboBox5.getSelectionModel().getSelectedIndex() < **0**) {

comboBox5.getSelectionModel().select(**0**);

}

**if** (comboBox6.getSelectionModel().getSelectedIndex() < **0**) {

comboBox6.getSelectionModel().select(**0**);

}

**if**(listFigures.get(comboBox5.getSelectionModel().getSelectedIndex()).cross(listFigures.get(comboBox6.getSelectionModel().getSelectedIndex()))){

mainController.label61.setVisible(**true**);

mainController.textBox41.setVisible(**true**);

mainController.label61.setText("Пересечение:");

mainController.textBox41.setText("Пересекаются.");

log.info("Фигура с координатами " + listFigures.get(comboBox5.getSelectionModel().getSelectedIndex()) + " и фигура с координатами " + listFigures.get(comboBox6.getSelectionModel().getSelectedIndex()) + " пересекаются" );

}

**else**{

mainController.label61.setVisible(**true**);

mainController.textBox41.setVisible(**true**);

mainController.label61.setText("Пересечение:");

mainController.textBox41.setText("Не пересекаются.");

log.info("Фигура с координатами " + listFigures.get(comboBox5.getSelectionModel().getSelectedIndex()) + " и фигура с координатами " + listFigures.get(comboBox6.getSelectionModel().getSelectedIndex()) + " НЕ пересекаются");

}

**if** (mainController.canvas != **null**) {

// Очищаем содержимое холста (canvas) с белым цветом

GraphicsContext gc = mainController.canvas.getGraphicsContext2D();

gc.setFill(Color.WHITE);

gc.clearRect(**0**, **0**, mainController.canvas.getWidth(), mainController.canvas.getHeight());

// Вызываем метод requestFocus(), чтобы перерисовать холст

mainController.canvas.requestFocus();

}

**assert** mainController.canvas != **null**;

application.drawAxes(mainController.canvas.getWidth(), mainController.canvas.getHeight());

**for**(**int** i = **0**; i < Storage.getListFigures().size(); i++){

mainController.draw(Storage.getListFigures().get(i), Color.BLACK);

}

mainController.draw(listFigures.get(comboBox5.getSelectionModel().getSelectedIndex()), Color.RED);

mainController.draw(listFigures.get(comboBox6.getSelectionModel().getSelectedIndex()), Color.RED);

Alert alert = **new** Alert(Alert.AlertType.INFORMATION);

alert.setTitle("Успех");

alert.setHeaderText(**null**);

alert.setContentText("Сравнение было выполнено успешно");

alert.showAndWait();

}

**catch** (EqualsException ex) {

Alert alert = **new** Alert(Alert.AlertType.WARNING);

alert.setTitle("Ошибка");

alert.setHeaderText(**null**);

alert.setContentText("Вы пытаетесь сравнить одну и ту же фигуру");

alert.showAndWait();

} **catch** (NullInputException ex) {

Alert alert = **new** Alert(Alert.AlertType.WARNING);

alert.setTitle("NullInputException");

alert.setHeaderText(**null**);

alert.setContentText(ex.toString());

alert.showAndWait();

} **catch** (IllegalArgumentException ex) { //неправильный ввод данных

Alert alert = **new** Alert(Alert.AlertType.WARNING);

alert.setTitle("IllegalArgumentException ex");

alert.setHeaderText(**null**);

alert.setContentText(ex.toString());

alert.showAndWait();

}

}

**public** **void** **CancelButton\_Click**(ActionEvent actionEvent) {

Stage stage = (Stage) ((Node) actionEvent.getSource()).getScene().getWindow();

// Закрыть текущую сцену (форму)

stage.close();

}

}

## **DeleteFiguresController**

**package** com.figures.figuresfx;

**import** **com.figures.exceptions.NullInputException**;

**import** **com.figures.storage.Storage**;

**import** **javafx.event.ActionEvent**;

**import** **javafx.fxml.FXML**;

**import** **javafx.scene.Node**;

**import** **javafx.scene.canvas.GraphicsContext**;

**import** **javafx.scene.control.Alert**;

**import** **javafx.scene.control.ComboBox**;

**import** **javafx.scene.paint.Color**;

**import** **javafx.stage.Stage**;

**import** **lombok.extern.java.Log**;

**import** **static** com.figures.storage.Storage.listFigures;

**import** **static** com.figures.storage.Storage.nameFigures;

**@Log**

**public** **class** **DeleteFiguresController** {

**@FXML**

**public** ComboBox<String> comboBox4;

**private** MainController mainController;

**private** AddFiguresController addFiguresController;

**private** CrossFiguresController crossFiguresController;

**private** MoveFiguresController moveFiguresController;

**public** **void** **setControllers**(MainController mainController, AddFiguresController addFiguresController, CrossFiguresController crossFiguresController, MoveFiguresController moveFiguresController) {

**this**.mainController = mainController;

**this**.addFiguresController = addFiguresController;

**this**.crossFiguresController = crossFiguresController;

**this**.moveFiguresController = moveFiguresController;

}

**private** FiguresApp application;

**public** **void** **setApplication**(FiguresApp application) {

**this**.application = application;

}

**public** **void** **initialize**(){

**for**(**int** i = **0**; i < Storage.getListFigures().size(); i++) {

String figure = Storage.nameFigures.get(i);

**switch** (figure) {

**case** "Окружность":

comboBox4.getItems().add("Окружность: " + "Circle" + Storage.getListFigures().get(i).toString());

**break**;

**case** "Отрезок":

comboBox4.getItems().add("Отрезок: " + "Segment" + Storage.getListFigures().get(i).toString());

**break**;

**case** "Ломанная": // Если фигура - полилиния

**case** "Многоугольник": // Если фигура - правильный n-угольник

**case** "Четырёхугольник": // Если фигура - квадрат

**case** "Треугольник": // Если фигура - равнобедренный треугольник

**case** "Трапеция": // Если фигура - трапеция

**case** "Прямоугольник": // Если фигура - прямоугольник

// Добавление фигуры в список в зависимости от типа

**if** (figure == "Ломанная") {

comboBox4.getItems().add("Ломанная: " + "Polyline" + Storage.getListFigures().get(i).toString());

}

**if** (figure == "Многоугольник") {

comboBox4.getItems().add("Многоугольник: " + "NGon" + Storage.getListFigures().get(i).toString());

}

**if** (figure == "Четырёхугольник") {

comboBox4.getItems().add("Четырёхугольник: " + "QGon" + Storage.getListFigures().get(i).toString());

}

**if** (figure == "Треугольник") {

comboBox4.getItems().add("Треугольник: " + "TGon" + Storage.getListFigures().get(i).toString());

}

**if** (figure == "Трапеция") {

comboBox4.getItems().add("Трапеция: " + "Trapeze" + Storage.getListFigures().get(i).toString());

}

**if** (figure == "Прямоугольник") {

comboBox4.getItems().add("Прямоугольник: " + "Rectangle" + Storage.getListFigures().get(i).toString());

}

**break**;

}

}

}

// Удалить

**@FXML**

**private** **void** **button6\_Click**(ActionEvent actionEvent){

**try** {

**if** (moveFiguresController.comboBox3.getSelectionModel().getSelectedIndex() == -**1** && Storage.getFiguresCount() == **0** || moveFiguresController.comboBox3.getSelectionModel().getSelectedIndex() == **0** && Storage.getFiguresCount() == **0**) {

**throw** **new** **NullInputException**();

}

**if** (moveFiguresController.comboBox3.getSelectionModel().getSelectedIndex() < **0**) {

moveFiguresController.comboBox3.getSelectionModel().select(**0**);

}

**if**(Storage.getFiguresCount() == **0**){

comboBox4.getSelectionModel().select(-**1**);

}

**else** **if** (comboBox4.getSelectionModel().getSelectedIndex() < **0**) {

comboBox4.getSelectionModel().select(**0**);

}

**if** (crossFiguresController.comboBox5.getSelectionModel().getSelectedIndex() < **0**) {

crossFiguresController.comboBox5.getSelectionModel().select(**0**);

}

**if** (crossFiguresController.comboBox6.getSelectionModel().getSelectedIndex() < **0**) {

crossFiguresController.comboBox6.getSelectionModel().select(**0**);

}

**if**(comboBox4.getSelectionModel().getSelectedIndex() != -**1**){

log.info("Фигура: " + nameFigures.get(comboBox4.getSelectionModel().getSelectedIndex()) + listFigures.get(comboBox4.getSelectionModel().getSelectedIndex()) + " была удалена");

Storage.getListFigures().remove(comboBox4.getSelectionModel().getSelectedIndex());

Storage.nameFigures.remove(comboBox4.getSelectionModel().getSelectedIndex());

comboBox4.getItems().remove(comboBox4.getSelectionModel().getSelectedIndex());

**if** (mainController.canvas != **null**) {

// Очищаем содержимое холста (canvas) с белым цветом

GraphicsContext gc = mainController.canvas.getGraphicsContext2D();

gc.setFill(Color.WHITE);

gc.clearRect(**0**, **0**, mainController.canvas.getWidth(), mainController.canvas.getHeight());

// Вызываем метод requestFocus(), чтобы перерисовать холст

mainController.canvas.requestFocus();

}

**assert** mainController.canvas != **null**;

application.drawAxes(mainController.canvas.getWidth(), mainController.canvas.getHeight());

**for**(**int** i = **0**; i < Storage.getListFigures().size(); i++){

mainController.draw(Storage.getListFigures().get(i), Color.BLACK);

}

Storage.decFiguresCount(); //Уменбшение кол-ва фигур

String type;

Storage.nameFigures.clear();

**for**(**int** j = **0**; j < Storage.figuresCount; j++ )

{

type = Storage.getListFigures().get(j).getClass().getSimpleName();

type = type.replace("FiguresFX.", "");

**if** ("Circle".equals(type)) {

Storage.nameFigures.add("Окружность");

} **else** **if** ("NGon".equals(type)) {

Storage.nameFigures.add("Многоугольник");

} **else** **if** ("Trapeze".equals(type)) {

Storage.nameFigures.add("Трапеция");

} **else** **if** ("Rectangle".equals(type)) {

Storage.nameFigures.add("Прямоугольник");

} **else** **if** ("TGon".equals(type)) {

Storage.nameFigures.add("Треугольник");

} **else** **if** ("Polyline".equals(type)) {

Storage.nameFigures.add("Ломанная");

} **else** **if** ("QGon".equals(type)) {

Storage.nameFigures.add("Четырёхугольник");

} **else** **if** ("Segment".equals(type)) {

Storage.nameFigures.add("Отрезок");

}

}

}

mainController.label61.setText("");

mainController.textBox41.setText("");

Alert alert = **new** Alert(Alert.AlertType.INFORMATION);

alert.setTitle("Успех");

alert.setHeaderText(**null**);

alert.setContentText("Удаление завершено");

alert.showAndWait();

}

**catch** (NullInputException ex) {

Alert alert = **new** Alert(Alert.AlertType.WARNING);

alert.setTitle("Ошибка");

alert.setHeaderText("NullInputException");

alert.setContentText(ex.toString());

alert.showAndWait();

} **catch** (IllegalArgumentException ex) {

Alert alert = **new** Alert(Alert.AlertType.WARNING);

alert.setTitle("Ошибка");

alert.setHeaderText("IllegalArgumentException ex");

alert.setContentText(ex.getMessage());

alert.showAndWait();

}

}

**public** **void** **CancelButton\_Click**(ActionEvent actionEvent) {

Stage stage = (Stage) ((Node) actionEvent.getSource()).getScene().getWindow();

// Закрыть текущую сцену (форму)

stage.close();

}

}

## **DoubleArrayCodec**

**package** com.figures.figuresfx;

**import** **org.bson.BsonReader**;

**import** **org.bson.BsonType**;

**import** **org.bson.BsonWriter**;

**import** **org.bson.codecs.Codec**;

**import** **org.bson.codecs.DecoderContext**;

**import** **org.bson.codecs.EncoderContext**;

**import** **java.util.ArrayList**;

**import** **java.util.List**;

**public** **class** **DoubleArrayCodec** **implements** Codec<**double**[]> {

**@Override**

**public** **void** **encode**(**final** BsonWriter writer, **final** **double**[] value, **final** EncoderContext encoderContext) {

writer.writeStartArray();

**for** (**double** d : value) {

writer.writeDouble(d);

}

writer.writeEndArray();

}

**@Override**

**public** **double**[] **decode**(**final** BsonReader reader, **final** DecoderContext decoderContext) {

reader.readStartArray();

List<Double> list = **new** ArrayList<>();

**while** (reader.readBsonType() != BsonType.END\_OF\_DOCUMENT) {

list.add(reader.readDouble());

}

reader.readEndArray();

**double**[] result = **new** **double**[list.size()];

**for** (**int** i = **0**; i < list.size(); i++) {

result[i] = list.get(i);

}

**return** result;

}

**@Override**

**public** Class<**double**[]> **getEncoderClass**() {

**return** **double**[].class;

}

}

## **FiguresApp**

**package** com.figures.figuresfx;

**import** **com.figures.storage.Storage**;

**import** **javafx.application.Application**;

**import** **javafx.fxml.FXMLLoader**;

**import** **javafx.scene.Scene**;

**import** **javafx.scene.canvas.Canvas**;

**import** **javafx.scene.canvas.GraphicsContext**;

**import** **javafx.scene.image.WritableImage**;

**import** **javafx.scene.layout.VBox**;

**import** **javafx.scene.paint.Color**;

**import** **javafx.stage.Stage**;

**import** **org.figures.interfaces.IShape**;

**import** **java.io.IOException**;

**import** **java.util.List**;

**import** **java.util.Objects**;

**public** **class** **FiguresApp** **extends** Application {

**private** WritableImage image;

**private** GraphicsContext graphicsContext;

**@Override**

**public** **void** **start**(Stage stage) **throws** IOException {

FXMLLoader fxmlLoadermain = **new** FXMLLoader(FiguresApp.class.getResource("main-view.fxml"));

FXMLLoader fxmlLoaderaddfigure = **new** FXMLLoader(FiguresApp.class.getResource("add\_figure-view.fxml"));

FXMLLoader fxmlLoadermovefigure = **new** FXMLLoader(FiguresApp.class.getResource("move\_figure-view.fxml"));

FXMLLoader fxmlLoaderdeletefigure = **new** FXMLLoader(FiguresApp.class.getResource("delete\_figure-view.fxml"));

FXMLLoader fxmlLoadercrossfigure = **new** FXMLLoader(FiguresApp.class.getResource("cross\_figure-view.fxml"));

FXMLLoader fxmlLoadersquarefigure = **new** FXMLLoader(FiguresApp.class.getResource("square\_figure-view.fxml"));

FXMLLoader fxmlLoaderperimetrfigure = **new** FXMLLoader(FiguresApp.class.getResource("perimeter\_figure-view.fxml"));

VBox root = fxmlLoadermain.load();

fxmlLoaderaddfigure.load();

fxmlLoadermovefigure.load();

fxmlLoaderdeletefigure.load();

fxmlLoadercrossfigure.load();

fxmlLoadersquarefigure.load();

fxmlLoaderperimetrfigure.load();

// Получаем контроллер

MainController controller = fxmlLoadermain.getController();

AddFiguresController controller1 = fxmlLoaderaddfigure.getController();

MoveFiguresController controller2 = fxmlLoadermovefigure.getController();

DeleteFiguresController controller3 = fxmlLoaderdeletefigure.getController();

CrossFiguresController controller4 = fxmlLoadercrossfigure.getController();

SquareFigureController controller5 = fxmlLoadersquarefigure.getController();

PerimeterFigureController controller6 = fxmlLoaderperimetrfigure.getController();

// Устанавливаем зависимость

controller.setControllers(controller4,controller1,controller2,controller3);

controller1.setControllers(controller,controller4, controller2,controller3);

controller2.setControllers(controller,controller1, controller4,controller3);

controller3.setControllers(controller,controller1, controller4,controller2);

controller4.setControllers(controller,controller1, controller2,controller3);

controller5.setControllers(controller);

controller6.setControllers(controller);

// Устанавливаем зависимость

controller.setApplication(**this**);

controller1.setApplication(**this**);

controller2.setApplication(**this**);

controller3.setApplication(**this**);

controller4.setApplication(**this**);

controller5.setApplication(**this**);

controller6.setApplication(**this**);

Scene scene = **new** Scene(root, **950**, **700**);

scene.getStylesheets().add(Objects.requireNonNull(getClass().getResource("styles.css")).toExternalForm());

stage.setTitle("FiguresApp");

stage.setScene(scene);

stage.show();

Canvas canvas = (Canvas) root.lookup("#canvas");

graphicsContext = canvas.getGraphicsContext2D();

image = **new** WritableImage((**int**) canvas.getWidth(), (**int**) canvas.getHeight());

graphicsContext = canvas.getGraphicsContext2D();

graphicsContext.drawImage(image, **0**, **0**);

// Draw X and Y axes

drawAxes(canvas.getWidth(), canvas.getHeight());

}

**public** **void** **drawAxes**(**double** canvasWidth, **double** canvasHeight) {

graphicsContext.setStroke(Color.GRAY);

graphicsContext.setLineWidth(**1**);

// X-axis

graphicsContext.strokeLine(**0**, canvasHeight / **2**, canvasWidth, canvasHeight / **2**);

// Y-axis

graphicsContext.strokeLine(canvasWidth / **2**, **0**, canvasWidth / **2**, canvasHeight);

}

**public** **void** **setFiguresCount**(**int** value) {

Storage.figuresCount = value;

}

**public** List<IShape> **getListFigures**(){

**return** Storage.listFigures;

}

**public** **static** **void** **main**(String[] args) {

launch();

}

}

## **MainController**

**package** com.figures.figuresfx;

**import** **com.mongodb.client.\***;

**import** **org.bson.codecs.configuration.CodecRegistries**;

**import** **com.figures.storage.Storage**;

**import** **com.mongodb.MongoClientSettings**;

**import** **javafx.event.ActionEvent**;

**import** **javafx.fxml.FXML**;

**import** **javafx.fxml.FXMLLoader**;

**import** **javafx.scene.Parent**;

**import** **javafx.scene.Scene**;

**import** **javafx.scene.SnapshotParameters**;

**import** **javafx.scene.canvas.Canvas**;

**import** **javafx.scene.canvas.GraphicsContext**;

**import** **javafx.scene.control.Alert**;

**import** **javafx.scene.control.Button**;

**import** **javafx.scene.control.Label**;

**import** **javafx.scene.control.TextField**;

**import** **javafx.scene.image.WritableImage**;

**import** **javafx.scene.paint.Color**;

**import** **javafx.stage.Modality**;

**import** **javafx.stage.Stage**;

**import** **org.bson.codecs.configuration.CodecRegistry**;

**import** **org.bson.codecs.pojo.PojoCodecProvider**;

**import** **org.figures.figures.\***;

**import** **org.figures.figures.points.Point**;

**import** **org.figures.figures.points.Point2D**;

**import** **org.figures.figures.points.Point3D**;

**import** **org.figures.interfaces.IShape**;

**import** **javax.imageio.ImageIO**;

**import** **java.awt.image.BufferedImage**;

**import** **java.io.\***;

**import** **java.util.ArrayList**;

**import** **java.util.List**;

**import** **java.util.regex.Matcher**;

**import** **java.util.regex.Pattern**;

**import** **org.bson.Document**;

**import** **com.mongodb.ConnectionString**;

**public** **class** **MainController**

{

**public** **MainController**(){}

**private** FiguresApp application;

**public** **void** **setApplication**(FiguresApp application) {

**this**.application = application;

}

**private** CrossFiguresController crossFiguresController;

**private** AddFiguresController addFiguresController;

**private** MoveFiguresController moveFiguresController;

**private** DeleteFiguresController deleteFiguresController;

**public** **void** **setControllers**(CrossFiguresController crossFiguresController, AddFiguresController addFiguresController, MoveFiguresController moveFiguresController, DeleteFiguresController deleteFiguresController) {

**this**.crossFiguresController = crossFiguresController;

**this**.addFiguresController = addFiguresController;

**this**.moveFiguresController = moveFiguresController;

**this**.deleteFiguresController = deleteFiguresController;

}

**@FXML**

**public** Canvas canvas;

**@FXML**

**public** Label label61;

**@FXML**

**public** TextField textBox41;

**@FXML**

**private** Button fbutton2;

**@FXML**

**private** Button fbutton3;

**@FXML**

**private** Button fbutton4;

**@FXML**

**private** Button fbutton5;

**@FXML**

**private** Button fbutton6;

**@FXML**

**private** Button fbutton7;

**@FXML**

**private** Button button\_loadINBD;

**@FXML**

**private** Button Button\_loadFromBD;

**public** **void** **initialize**() {}

//Очиска холста

**@FXML**

**private** **void** **button1\_Click**(ActionEvent actionEvent){

**if** (canvas != **null**) {

// Очищаем содержимое холста (canvas) с белым цветом

GraphicsContext gc = canvas.getGraphicsContext2D();

gc.setFill(Color.WHITE);

gc.clearRect(**0**, **0**, canvas.getWidth(), canvas.getHeight());

// Вызываем метод requestFocus(), чтобы перерисовать холст

canvas.requestFocus();

}

**assert** canvas != **null**;

application.drawAxes(canvas.getWidth(), canvas.getHeight());

application.getListFigures().clear();

application.setFiguresCount(**0**);

moveFiguresController.comboBox3.getItems().clear();

deleteFiguresController.comboBox4.getItems().clear();

crossFiguresController.comboBox5.getItems().clear();

crossFiguresController.comboBox6.getItems().clear();

label61.setText("");

textBox41.setText("");

Alert alert = **new** Alert(Alert.AlertType.INFORMATION);

alert.setTitle("Успех");

alert.setHeaderText(**null**);

alert.setContentText("Все фигуры удалены");

alert.showAndWait();

}

//Форма с площадью

**@FXML**

**private** **void** **fbutton2\_Click**(ActionEvent actionEvent){

// Проверяем, что listFigures не пустой

**if** (Storage.getListFigures().isEmpty()) {

// Выводим Alert, если listFigures пустой

Alert alert = **new** Alert(Alert.AlertType.ERROR);

alert.setTitle("Ошибка");

alert.setHeaderText(**null**);

alert.setContentText("Добавьте хотя бы одну фигуру");

alert.showAndWait();

**return**; // Прекращаем выполнение метода, так как нет фигур для сохранения

}

**try** {

FXMLLoader loader = **new** FXMLLoader(getClass().getResource("square\_figure-view.fxml"));

Parent root = loader.load();

SquareFigureController controller1 = loader.getController();

controller1.setControllers(**this**);

controller1.setApplication(application);

// Создайте новое модальное окно

Stage moveFigureStage = **new** Stage();

moveFigureStage.initModality(Modality.APPLICATION\_MODAL); // Сделать окно модальным

moveFigureStage.setTitle("Площадь фигуры");

moveFigureStage.setScene(**new** Scene(root));

// Заблокируйте взаимодействие с главным окном

moveFigureStage.initOwner(fbutton2.getScene().getWindow());

// Отобразите окно и дождитесь его закрытия

moveFigureStage.showAndWait();

} **catch** (IOException e) {

e.printStackTrace();

}

}

// Периметр

**public** **void** **fbutton3\_Click**(ActionEvent actionEvent) {

// Проверяем, что listFigures не пустой

**if** (Storage.getListFigures().isEmpty()) {

// Выводим Alert, если listFigures пустой

Alert alert = **new** Alert(Alert.AlertType.ERROR);

alert.setTitle("Ошибка");

alert.setHeaderText(**null**);

alert.setContentText("Добавьте хотя бы одну фигуру");

alert.showAndWait();

**return**; // Прекращаем выполнение метода, так как нет фигур для сохранения

}

**try** {

//загрузка сцены

FXMLLoader loader = **new** FXMLLoader(getClass().getResource("perimeter\_figure-view.fxml"));

Parent root = loader.load();

PerimeterFigureController controller1 = loader.getController();

controller1.setControllers(**this**);

controller1.setApplication(application);

// Создайте новое модальное окно

Stage moveFigureStage = **new** Stage();

moveFigureStage.initModality(Modality.APPLICATION\_MODAL); // Сделать окно модальным

moveFigureStage.setTitle("Периметр фигуры");

moveFigureStage.setScene(**new** Scene(root));

// Заблокируйте взаимодействие с главным окном

moveFigureStage.initOwner(fbutton3.getScene().getWindow());

// Отобразите окно и дождитесь его закрытия

moveFigureStage.showAndWait();

} **catch** (IOException e) {

e.printStackTrace();

}

}

//Добавление фигуры

**@FXML**

**public** **void** **fbutton4\_click**() {

**try** {

FXMLLoader loader = **new** FXMLLoader(getClass().getResource("add\_figure-view.fxml"));

Parent root = loader.load();

AddFiguresController controller1 = loader.getController();

FXMLLoader loader2 = **new** FXMLLoader(getClass().getResource("move\_figure-view.fxml"));

loader2.load();

MoveFiguresController controller2 = loader2.getController();

FXMLLoader loader3 = **new** FXMLLoader(getClass().getResource("delete\_figure-view.fxml"));

loader3.load();

DeleteFiguresController controller3 = loader3.getController();

FXMLLoader loader4 = **new** FXMLLoader(getClass().getResource("cross\_figure-view.fxml"));

loader4.load();

CrossFiguresController controller4 = loader4.getController();

controller1.setControllers(**this**, controller4, controller2, controller3);

controller1.setApplication(application);

controller2.setApplication(application);

controller3.setApplication(application);

controller4.setApplication(application);

// Создайте новое модальное окно

Stage moveFigureStage = **new** Stage();

moveFigureStage.initModality(Modality.APPLICATION\_MODAL); // Сделать окно модальным

moveFigureStage.setTitle("Добавление фигуры");

moveFigureStage.setScene(**new** Scene(root));

// Заблокируйте взаимодействие с главным окном

moveFigureStage.initOwner(fbutton4.getScene().getWindow());

// Отобразите окно и дождитесь его закрытия

moveFigureStage.showAndWait();

} **catch** (IOException e) {

e.printStackTrace();

}

}

// Перемещение

**@FXML**

**public** **void** **fbutton5\_click**(ActionEvent actionEvent) {

**try** {

FXMLLoader loader = **new** FXMLLoader(getClass().getResource("add\_figure-view.fxml"));

loader.load();

AddFiguresController controller1 = loader.getController();

FXMLLoader loader2 = **new** FXMLLoader(getClass().getResource("move\_figure-view.fxml"));

Parent root = loader2.load();

MoveFiguresController controller2 = loader2.getController();

FXMLLoader loader3 = **new** FXMLLoader(getClass().getResource("delete\_figure-view.fxml"));

loader3.load();

DeleteFiguresController controller3 = loader3.getController();

FXMLLoader loader4 = **new** FXMLLoader(getClass().getResource("cross\_figure-view.fxml"));

loader4.load();

CrossFiguresController controller4 = loader4.getController();

controller2.setControllers(**this**, controller1, controller4, controller3);

controller1.setApplication(application);

controller2.setApplication(application);

controller3.setApplication(application);

controller4.setApplication(application);

// Создайте новое модальное окно

Stage moveFigureStage = **new** Stage();

moveFigureStage.initModality(Modality.APPLICATION\_MODAL); // Сделать окно модальным

moveFigureStage.setTitle("Перемещение фигуры");

moveFigureStage.setScene(**new** Scene(root));

// Заблокируйте взаимодействие с главным окном

moveFigureStage.initOwner(fbutton5.getScene().getWindow());

// Отобразите окно и дождитесь его закрытия

moveFigureStage.showAndWait();

} **catch** (IOException e) {

e.printStackTrace();

}

}

//Удаление

**@FXML**

**public** **void** **fbutton6\_click**(ActionEvent actionEvent) {

**try** {

FXMLLoader loader = **new** FXMLLoader(getClass().getResource("add\_figure-view.fxml"));

loader.load();

AddFiguresController controller1 = loader.getController();

FXMLLoader loader2 = **new** FXMLLoader(getClass().getResource("move\_figure-view.fxml"));

loader2.load();

MoveFiguresController controller2 = loader2.getController();

FXMLLoader loader3 = **new** FXMLLoader(getClass().getResource("delete\_figure-view.fxml"));

Parent root = loader3.load();

DeleteFiguresController controller3 = loader3.getController();

FXMLLoader loader4 = **new** FXMLLoader(getClass().getResource("cross\_figure-view.fxml"));

loader4.load();

CrossFiguresController controller4 = loader4.getController();

controller3.setControllers(**this**, controller1, controller4, controller2);

controller1.setApplication(application);

controller2.setApplication(application);

controller3.setApplication(application);

controller4.setApplication(application);

// Создайте новое модальное окно

Stage moveFigureStage = **new** Stage();

moveFigureStage.initModality(Modality.APPLICATION\_MODAL); // Сделать окно модальным

moveFigureStage.setTitle("Удаление фигуры");

moveFigureStage.setScene(**new** Scene(root));

// Заблокируйте взаимодействие с главным окном

moveFigureStage.initOwner(fbutton6.getScene().getWindow());

// Отобразите окно и дождитесь его закрытия

moveFigureStage.showAndWait();

} **catch** (IOException e) {

e.printStackTrace();

}

}

//Пересечение

**@FXML**

**public** **void** **fbutton7\_click**(ActionEvent actionEvent) {

**try** {

FXMLLoader loader = **new** FXMLLoader(getClass().getResource("add\_figure-view.fxml"));

loader.load();

AddFiguresController controller1 = loader.getController();

FXMLLoader loader2 = **new** FXMLLoader(getClass().getResource("move\_figure-view.fxml"));

loader2.load();

MoveFiguresController controller2 = loader2.getController();

FXMLLoader loader3 = **new** FXMLLoader(getClass().getResource("delete\_figure-view.fxml"));

loader3.load();

DeleteFiguresController controller3 = loader3.getController();

FXMLLoader loader4 = **new** FXMLLoader(getClass().getResource("cross\_figure-view.fxml"));

Parent root = loader4.load();

CrossFiguresController controller4 = loader4.getController();

controller4.setControllers(**this**, controller1, controller2, controller3);

controller1.setApplication(application);

controller2.setApplication(application);

controller3.setApplication(application);

controller4.setApplication(application);

// Создайте новое модальное окно

Stage moveFigureStage = **new** Stage();

moveFigureStage.initModality(Modality.APPLICATION\_MODAL); // Сделать окно модальным

moveFigureStage.setTitle("Пересечение фигур");

moveFigureStage.setScene(**new** Scene(root));

// Заблокируйте взаимодействие с главным окном

moveFigureStage.initOwner(fbutton7.getScene().getWindow());

// Отобразите окно и дождитесь его закрытия

moveFigureStage.showAndWait();

} **catch** (IOException e) {

e.printStackTrace();

}

}

//Отрисовка

**public** **void** **draw**(IShape i, Color color) {

**double**[] mas\_x = **new** **double**[**0**];

**double**[] mas\_y = **new** **double**[**0**];

GraphicsContext gc = canvas.getGraphicsContext2D();

**if** (i **instanceof** Circle) {

mas\_x = **new** **double**[**101**];

mas\_y = **new** **double**[**101**];

Circle f = (Circle) i;

**for** (**int** j = **0**; j <= **100**; j++) {

**double** x = f.getP().getX(**0**) + f.getR() \* Math.cos(j \* **2** \* Math.PI / **100**);

**double** y = f.getP().getX(**1**) + f.getR() \* Math.sin(j \* **2** \* Math.PI / **100**);

mas\_x[j] = x;

mas\_y[j] = y;

}

} **else** **if** (i **instanceof** Segment) {

mas\_x = **new** **double**[**2**];

mas\_y = **new** **double**[**2**];

Segment f = (Segment) i;

mas\_x[**0**] = f.getStart().getX(**0**);

mas\_y[**0**] = f.getStart().getX(**1**);

mas\_x[**1**] = f.getFinish().getX(**0**);

mas\_y[**1**] = f.getFinish().getX(**1**);

} **else** **if** (i **instanceof** Polyline) {

**int** q = **0**;

Polyline f = (Polyline) i;

Point2D[] pointss = f.getP();

mas\_x = **new** **double**[pointss.length];

mas\_y = **new** **double**[pointss.length];

**for** (**int** j = **0**; j < pointss.length; j++) {

Point2D v = pointss[j];

mas\_x[q] = v.getX(**0**);

mas\_y[q] = v.getX(**1**);

q++;

}

} **else** **if** (i **instanceof** NGon) {

**int** q = **0**;

NGon f = (NGon) i;

mas\_x = **new** **double**[f.getP().length + **1**];

mas\_y = **new** **double**[f.getP().length + **1**];

**for** (Point2D v : f.getP()) {

mas\_x[q] = v.getX(**0**);

mas\_y[q] = v.getX(**1**);

q++;

}

mas\_x[f.getP().length] = f.getP()[**0**].getX(**0**);

mas\_y[f.getP().length] = f.getP()[**0**].getX(**1**);

}

// Коэффициент масштабирования

**double** scaleFactor = **8.0**;

// Получаем размеры Canvas

**double** canvasWidth = canvas.getWidth();

**double** canvasHeight = canvas.getHeight();

gc.setStroke(color);

Scale(i, mas\_x, mas\_y, gc, scaleFactor, canvasWidth, canvasHeight);

}

**private** **void** **Scale**(IShape i, **double**[] mas\_x, **double**[] mas\_y, GraphicsContext gc, **double** scaleFactor, **double** canvasWidth, **double** canvasHeight) {

**for** (**int** q = **0**; q < mas\_x.length - **1**; q++) {

**double** x1 = mas\_x[q] \* scaleFactor + canvasWidth / **2**;

**double** y1 = -mas\_y[q] \* scaleFactor + canvasHeight / **2**;

**double** x2 = mas\_x[q + **1**] \* scaleFactor + canvasWidth / **2**;

**double** y2 = -mas\_y[q + **1**] \* scaleFactor + canvasHeight / **2**;

gc.strokeLine(x1, y1, x2, y2);

}

// чтобы не замыкать ломанную

**if** (i **instanceof** Polyline) {

// Дополнительные действия для Polyline, если нужно

} **else** {

**double** x1 = mas\_x[mas\_x.length - **1**] \* scaleFactor + canvasWidth / **2**;

**double** y1 = -mas\_y[mas\_y.length - **1**] \* scaleFactor + canvasHeight / **2**;

**double** x2 = mas\_x[**0**] \* scaleFactor + canvasWidth / **2**;

**double** y2 = -mas\_y[**0**] \* scaleFactor + canvasHeight / **2**;

gc.strokeLine(x1, y1, x2, y2);

}

}

//Сохранение

**@FXML**

**public** **void** **button\_save\_Click**(ActionEvent actionEvent) {

// Проверяем, что список фигур не пуст

**if** (Storage.getListFigures().isEmpty()) {

// Если список пуст, выводим предупреждение и завершаем метод

showAlert("Ошибка", "Вы еще не добавили ни одной фигуры.");

**return**;

}

// Задайте относительный путь и имя файла в директории проекта

String relativePath = "save\_figures.txt"; // Имя файла и относительный путь

// Создаем файл в указанном пути

File file = **new** File(relativePath);

**try** (BufferedWriter writer = **new** BufferedWriter(**new** FileWriter(file))) {

writer.write(String.valueOf(Storage.figuresCount));

writer.newLine();

// Записываем каждую фигуру в файл на отдельной строке

**for** (**int** i = **0**; i < Storage.listFigures.size(); i++) {

writer.write(getFigureType(Storage.listFigures.get(i)));

writer.write(" ");

writer.write(formatFigure(Storage.listFigures.get(i)));

writer.newLine(); // Переходим на новую строку

}

showAlert("Успех", "Фигуры успешно сохранены в файл: " + file.getCanonicalPath());

} **catch** (IOException e) {

e.printStackTrace();

showAlert("Ошибка", "Произошла ошибка при сохранении данных.");

}

}

**private** String **formatFigure**(IShape figure) {

String figureString = figure.toString();

**if** (figure **instanceof** Circle || figure **instanceof** Segment) {

// Special handling for Circle and Segment

List<Double> numbers = extractNumbersFromString(figureString);

// Build the formatted string without including vertex count

StringBuilder formattedFigure = **new** StringBuilder();

**for** (Double number : numbers) {

formattedFigure.append(number.intValue()).append(" ");

}

**return** formattedFigure.toString().trim();

} **else** {

// For other figures with vertices

// Check if the figureString contains numbers

**boolean** hasNumbers = figureString.matches(".\*\\d+.\*");

**if** (hasNumbers) {

// Extract numbers from the toString output

List<Double> numbers = extractNumbersFromString(figureString);

// Build the formatted string

StringBuilder formattedFigure = **new** StringBuilder();

formattedFigure.append(numbers.size() / **2**).append(" ");

**for** (Double number : numbers) {

formattedFigure.append(number.intValue()).append(" ");

}

**return** formattedFigure.toString().trim();

} **else** {

// If the toString output doesn't contain numbers, return an empty string or handle as needed

**return** "";

}

}

}

**private** List<Double> **extractNumbersFromString**(String input) {

List<Double> numbers = **new** ArrayList<>();

Matcher matcher = Pattern.compile("-?\\d+(\\.\\d+)?").matcher(input);

**while** (matcher.find()) {

numbers.add(Double.parseDouble(matcher.group()));

}

**return** numbers;

}

//Получение типа фигуры

**private** String **getFigureType**(IShape figure) {

**if** (figure **instanceof** Circle) {

**return** "Circle";

} **else** **if** (figure **instanceof** Segment) {

**return** "Segment";

} **else** **if** (figure **instanceof** Polyline) {

**return** "Polyline";

}**else** **if** (figure **instanceof** Rectangle) {

**return** "Rectangle";

} **else** **if** (figure **instanceof** Trapeze) {

**return** "Trapeze";

} **else** **if** (figure **instanceof** TGon) {

**return** "TGon";

} **else** **if** (figure **instanceof** QGon) {

**return** "QGon";

}**else** **if** (figure **instanceof** NGon) {

**return** "NGon";

} **else** {

// Handle unknown figure types or add more conditions as needed

**return** "Unknown";

}

}

//Загрузка из файла

**@FXML**

**public** **void** **button\_loadFile\_Click**(ActionEvent actionEvent) {

**try** {

Storage.listFigures.clear();

Storage.nameFigures.clear();

BufferedReader reader = **new** BufferedReader(**new** FileReader("load\_figures.txt"));

String n = reader.readLine();

Storage.figuresCount = Integer.parseInt(n);

String line;

**while** ((line = reader.readLine()) != **null**) {

String[] values = line.split(" ");

String figure = values[**0**];

**switch** (figure) {

**case** "Circle":

Storage.nameFigures.add("Окружность");

Storage.listFigures.add(**new** Circle(**new** Point2D(**new** **double**[]{Double.parseDouble(values[**1**]), Double.parseDouble(values[**2**])}), Double.parseDouble(values[**3**])));

**break**;

**case** "Segment":

Storage.nameFigures.add("Отрезок");

Storage.listFigures.add(**new** Segment(**new** Point2D(**new** **double**[]{Double.parseDouble(values[**1**]), Double.parseDouble(values[**2**])}),

**new** **Point2D**(**new** **double**[]{Double.parseDouble(values[**3**]), Double.parseDouble(values[**4**])})));

**break**;

**case** "Polyline":

**case** "NGon":

**case** "QGon":

**case** "TGon":

**case** "Trapeze":

**case** "Rectangle":

**int** k = Integer.parseInt(values[**1**]);

Point2D[] pr = **new** Point2D[k];

**for** (**int** i = **0**, j = **2**; i < k; i++, j += **2**) {

pr[i] = **new** Point2D(**new** **double**[]{Double.parseDouble(values[j]), Double.parseDouble(values[j + **1**])});

}

**if** (figure.equals("Polyline"))

{

Storage.nameFigures.add("Ломанная");

Storage.listFigures.add(**new** Polyline(pr));

}

**if** (figure.equals("NGon"))

{

Storage.nameFigures.add("Многоугольник");

Storage.listFigures.add(**new** NGon(pr));

}

**if** (figure.equals("QGon"))

{

Storage.nameFigures.add("Четырёхугольник");

Storage.listFigures.add(**new** QGon(pr));

}

**if** (figure.equals("TGon"))

{

Storage.nameFigures.add("Треугольник");

Storage.listFigures.add(**new** TGon(pr));

}

**if** (figure.equals("Trapeze"))

{

Storage.nameFigures.add("Трапеция");

Storage.listFigures.add(**new** Trapeze(pr));

}

**if** (figure.equals("Rectangle"))

{

Storage.nameFigures.add("Прямоугольник");

Storage.listFigures.add(**new** Rectangle(pr));

}

**break**;

**default**:

System.out.println("Введён неверный тип фигуры: " + figure);

**break**;

}

}

reader.close();

GraphicsContext gc = canvas.getGraphicsContext2D();

gc.clearRect(**0**, **0**, canvas.getWidth(), canvas.getHeight());

application.drawAxes(canvas.getWidth(), canvas.getHeight());

**for**(**int** i = **0**; i < Storage.figuresCount; i++){

draw(Storage.getListFigures().get(i), Color.BLACK);

}

showAlert("Успех", "Фигуры успешно загружены");

} **catch** (IOException ex) {

showAlert("Ошибка", "Произошла ошибка при сохранении данных.");

}

}

//Сохранение картинки

**@FXML**

**public** **void** **button\_saveIMG\_Click**(ActionEvent actionEvent) {

// Проверяем, что listFigures не пустой

**if** (Storage.getListFigures().isEmpty()) {

// Выводим Alert, если listFigures пустой

Alert alert = **new** Alert(Alert.AlertType.ERROR);

alert.setTitle("Ошибка");

alert.setHeaderText(**null**);

alert.setContentText("Добавьте хотя бы одну фигуру");

alert.showAndWait();

**return**; // Прекращаем выполнение метода, так как нет фигур для сохранения

}

// Создаем объект класса WritableImage, чтобы сохранить содержимое Canvas в изображении

**int** width = (**int**) canvas.getWidth();

**int** height = (**int**) canvas.getHeight();

WritableImage writableImage = **new** WritableImage(width, height);

SnapshotParameters parameters = **new** SnapshotParameters();

canvas.snapshot(parameters, writableImage);

// Указываем путь и имя файла для сохранения

String fileName = "saved\_image.png";

File file = **new** File(fileName);

**try** {

// Создаем BufferedImage из WritableImage

BufferedImage bufferedImage = **new** BufferedImage(width, height, BufferedImage.TYPE\_INT\_ARGB);

**for** (**int** x = **0**; x < width; x++) {

**for** (**int** y = **0**; y < height; y++) {

bufferedImage.setRGB(x, y, writableImage.getPixelReader().getArgb(x, y));

}

}

// Сохраняем изображение

ImageIO.write(bufferedImage, "png", file);

// Выводим сообщение об успешном сохранении

Alert alert = **new** Alert(Alert.AlertType.INFORMATION);

alert.setTitle("Сохранено");

alert.setHeaderText(**null**);

alert.setContentText("Изображение успешно сохранено в " + fileName);

alert.showAndWait();

} **catch** (IOException e) {

e.printStackTrace();

// В случае ошибки сохранения выводим сообщение об ошибке

Alert alert = **new** Alert(Alert.AlertType.ERROR);

alert.setTitle("Ошибка");

alert.setHeaderText(**null**);

alert.setContentText("Произошла ошибка при сохранении изображения");

alert.showAndWait();

}

}

**private** **void** **showAlert**(String title, String content) {

Alert alert = **new** Alert(Alert.AlertType.INFORMATION);

alert.setTitle(title);

alert.setHeaderText(**null**);

alert.setContentText(content);

alert.showAndWait();

}

//Загрузка в базу данных

**public** **void** **button\_loadINBD\_Click**(ActionEvent actionEvent) {

**try** {

CodecRegistry codecRegistry = CodecRegistries.fromRegistries(

MongoClientSettings.getDefaultCodecRegistry(),

CodecRegistries.fromCodecs(**new** DoubleArrayCodec())

);

PojoCodecProvider pojoCodecProvider = PojoCodecProvider.builder().automatic(**true**).build();

CodecRegistry pojoCodecRegistry = CodecRegistries.fromProviders(pojoCodecProvider);

CodecRegistry combinedCodecRegistry = CodecRegistries.fromRegistries(codecRegistry, pojoCodecRegistry);

String connectionString = "mongodb://localhost:27017";

ConnectionString connString = **new** ConnectionString(connectionString);

MongoClientSettings settings = MongoClientSettings.builder()

.applyConnectionString(connString)

.codecRegistry(combinedCodecRegistry)

.build();

**try** (MongoClient mongoClient = MongoClients.create(settings)) {

MongoDatabase database = mongoClient.getDatabase("Figures");

// Очистка коллекции перед добавлением новых фигур

MongoCollection<Document> collection = database.getCollection("Figures");

collection.deleteMany(**new** Document());

// Цикл по всем фигурам

**for** (**int** i = **0**; i < Storage.listFigures.size(); i++) {

IShape figure = Storage.listFigures.get(i);

Document document = createFigureDocument(figure);

collection.insertOne(document);

}

}

} **catch** (Exception e) {

e.printStackTrace();

}

}

// Загрузка из БД

**public** **void** **Button\_loadFromBD\_Click**(ActionEvent actionEvent) {

**try** {

Storage.listFigures.clear();

Storage.nameFigures.clear();

Storage.figuresCount = **0**;

CodecRegistry codecRegistry = CodecRegistries.fromRegistries(

MongoClientSettings.getDefaultCodecRegistry(),

CodecRegistries.fromCodecs(**new** DoubleArrayCodec())

);

PojoCodecProvider pojoCodecProvider = PojoCodecProvider.builder().automatic(**true**).build();

CodecRegistry pojoCodecRegistry = CodecRegistries.fromProviders(pojoCodecProvider);

CodecRegistry combinedCodecRegistry = CodecRegistries.fromRegistries(codecRegistry, pojoCodecRegistry);

String connectionString = "mongodb://localhost:27017";

ConnectionString connString = **new** ConnectionString(connectionString);

MongoClientSettings settings = MongoClientSettings.builder()

.applyConnectionString(connString)

.codecRegistry(combinedCodecRegistry)

.build();

**try** (MongoClient mongoClient = MongoClients.create(settings)) {

MongoDatabase database = mongoClient.getDatabase("Figures");

// Получаем коллекцию

MongoCollection<Document> collection = database.getCollection("Figures");

// Находим все документы в коллекции

FindIterable<Document> documents = collection.find();

// Очищаем список фигур перед загрузкой новых

Storage.listFigures.clear();

// Проходим по документам и создаем объекты фигур

**for** (Document document : documents) {

IShape figure = createFigureFromDocument(document);

// Добавляем фигуру в список фигур

Storage.listFigures.add(figure);

Storage.figuresCount++;

}

}

GraphicsContext gc = canvas.getGraphicsContext2D();

gc.clearRect(**0**, **0**, canvas.getWidth(), canvas.getHeight());

application.drawAxes(canvas.getWidth(), canvas.getHeight());

**for**(**int** i = **0**; i < Storage.figuresCount; i++){

draw(Storage.getListFigures().get(i), Color.BLACK);

}

} **catch** (Exception e) {

e.printStackTrace();

}

}

//Создание фигуры из json

**private** IShape **createFigureFromDocument**(Document document) {

String figureType = document.getString("type");

**switch** (figureType) {

**case** "Circle":

Storage.nameFigures.add("Окружность");

**return** **createCircleFromDocument**(document);

**case** "Segment":

Storage.nameFigures.add("Отрезок");

**return** **createSegmentFromDocument**(document);

**case** "Polyline":

Storage.nameFigures.add("Ломанная");

**return** **createPolylineFromDocument**(document);

**case** "Rectangle":

Storage.nameFigures.add("Прямоугольник");

**return** **createRectangleFromDocument**(document);

**case** "TGon":

Storage.nameFigures.add("Треугольник");

**return** **createTGonFromDocument**(document);

**case** "Trapeze":

Storage.nameFigures.add("Трапеция");

**return** **createTrapezeFromDocument**(document);

**case** "QGon":

Storage.nameFigures.add("Четырёхугольник");

**return** **createQGonFromDocument**(document);

**case** "NGon":

Storage.nameFigures.add("Многоугольник");

**return** **createNGonFromDocument**(document);

**default**:

**throw** **new** **IllegalArgumentException**("Неизвестный тип фигуры: " + figureType);

}

}

**private** Circle **createCircleFromDocument**(Document document) {

**double** radius = document.getDouble("r");

Document centerDocument = document.get("p", Document.class);

Point2D center = createPoint2DFromDocument(centerDocument);

**return** **new** **Circle**(center, radius);

}

**private** Segment **createSegmentFromDocument**(Document document) {

Document startDocument = document.get("start", Document.class);

Document finishDocument = document.get("finish", Document.class);

Point2D start = createPoint2DFromDocument(startDocument);

Point2D finish = createPoint2DFromDocument(finishDocument);

**return** **new** **Segment**(start, finish);

}

**private** Polyline **createPolylineFromDocument**(Document document) {

**int** n = document.getInteger("n");

List<Document> pointsDocuments = document.getList("points", Document.class);

Point2D[] points = pointsDocuments.stream()

.map(**this**::createPoint2DFromDocument)

.toArray(Point2D[]::**new**);

**return** **new** **Polyline**(points);

}

**private** Rectangle **createRectangleFromDocument**(Document document) {

**int** n = document.getInteger("n");

List<Document> pointsDocuments = document.getList("points", Document.class);

Point2D[] points = pointsDocuments.stream()

.map(**this**::createPoint2DFromDocument)

.toArray(Point2D[]::**new**);

**return** **new** **Rectangle**(points);

}

**private** NGon **createNGonFromDocument**(Document document) {

**int** n = document.getInteger("n");

List<Document> pointsDocuments = document.getList("points", Document.class);

Point2D[] points = pointsDocuments.stream()

.map(**this**::createPoint2DFromDocument)

.toArray(Point2D[]::**new**);

**return** **new** **NGon**(points);

}

**private** TGon **createTGonFromDocument**(Document document) {

**int** n = document.getInteger("n");

List<Document> pointsDocuments = document.getList("points", Document.class);

Point2D[] points = pointsDocuments.stream()

.map(**this**::createPoint2DFromDocument)

.toArray(Point2D[]::**new**);

**return** **new** **TGon**(points);

}

**private** Trapeze **createTrapezeFromDocument**(Document document) {

**int** n = document.getInteger("n");

List<Document> pointsDocuments = document.getList("points", Document.class);

Point2D[] points = pointsDocuments.stream()

.map(**this**::createPoint2DFromDocument)

.toArray(Point2D[]::**new**);

**return** **new** **Trapeze**(points);

}

**private** QGon **createQGonFromDocument**(Document document) {

**int** n = document.getInteger("n");

List<Document> pointsDocuments = document.getList("points", Document.class);

Point2D[] points = pointsDocuments.stream()

.map(**this**::createPoint2DFromDocument)

.toArray(Point2D[]::**new**);

**return** **new** **QGon**(points);

}

**private** Point2D **createPoint2DFromDocument**(Document document) {

List<Double> coordinates = document.getList("x", Double.class);

**double**[] coordsArray = coordinates.stream().mapToDouble(**Double:**:doubleValue).toArray();

**return** **new** **Point2D**(coordsArray);

}

// Метод для создания документа для различных типов фигур

**private** Document **createFigureDocument**(IShape figure) {

Document document = **new** Document();

**if** (figure **instanceof** Circle) {

document = createCircleDocument((Circle) figure);

} **else** **if** (figure **instanceof** Segment) {

document = createSegmentDocument((Segment) figure);

} **else** **if** (figure **instanceof** Polyline) {

document = createPolylineDocument((Polyline) figure);

} **else** **if** (figure **instanceof** Rectangle) {

document = createRectangleDocument((Rectangle) figure);

} **else** **if** (figure **instanceof** TGon) {

document = createTGonDocument((TGon) figure);

} **else** **if** (figure **instanceof** Trapeze) {

document = createTrapezeDocument((Trapeze) figure);

} **else** **if** (figure **instanceof** QGon) {

document = createQGonDocument((QGon) figure);

} **else** **if** (figure **instanceof** NGon) {

document = createNGonDocument((NGon) figure);

}

// Добавьте обработку других типов фигур

**return** document;

}

// Методы для создания документов для различных типов фигур

**private** Document **createPointDocument**(Point point) {

Document document = **new** Document();

document.append("type", "Point");

document.append("dim", point.getDim());

document.append("x", point.getX());

// Добавляем точки вершин, если есть

//document.append("points", createPointDocuments(point.getP()));

**return** document;

}

**private** Document **createPoint2DDocument**(Point2D point) {

Document document = **new** Document();

document.append("type", "Point2D");

document.append("dim", point.getDim());

document.append("x", point.getX());

// Добавляем точки вершин, если есть

//document.append("points", createPointDocuments(point.getP()));

**return** document;

}

**private** Document **createPoint3DDocument**(Point3D point) {

Document document = **new** Document();

document.append("type", "Point3D");

document.append("dim", point.getDim());

document.append("x", point.getX());

**return** document;

}

**private** Document **createCircleDocument**(Circle circle) {

Document document = **new** Document();

document.append("type", "Circle");

document.append("r", circle.getR());

document.append("p", createPointDocument(circle.getP()));

**return** document;

}

**private** Document **createNGonDocument**(NGon nGon) {

Document document = **new** Document();

document.append("type", "NGon");

document.append("n", nGon.getN());

document.append("points", createPointDocuments(nGon.getP()));

**return** document;

}

**private** Document **createPolylineDocument**(Polyline polyline) {

Document document = **new** Document();

document.append("type", "Polyline");

document.append("n", polyline.getN());

document.append("points", createPointDocuments(polyline.getP()));

**return** document;

}

**private** Document **createQGonDocument**(QGon qGon) {

Document document = **new** Document();

document.append("type", "QGon");

document.append("n", qGon.getN());

document.append("points", createPointDocuments(qGon.getP()));

**return** document;

}

**private** Document **createRectangleDocument**(Rectangle rectangle) {

Document document = **new** Document();

document.append("type", "Rectangle");

document.append("n", rectangle.getN());

document.append("points", createPointDocuments(rectangle.getP()));

**return** document;

}

**private** Document **createSegmentDocument**(Segment segment) {

Document document = **new** Document();

document.append("type", "Segment");

document.append("start", createPointDocument(segment.getStart()));

document.append("finish", createPointDocument(segment.getFinish()));

**return** document;

}

**private** Document **createTGonDocument**(TGon tGon) {

Document document = **new** Document();

document.append("type", "TGon");

document.append("n", tGon.getN());

document.append("points", createPointDocuments(tGon.getP()));

**return** document;

}

**private** Document **createTrapezeDocument**(Trapeze trapeze) {

Document document = **new** Document();

document.append("type", "Trapeze");

document.append("n", trapeze.getN());

document.append("points", createPointDocuments(trapeze.getP()));

**return** document;

}

**private** List<Document> **createPointDocuments**(Point2D[] points) {

List<Document> pointDocuments = **new** ArrayList<>();

**for** (Point point : points) {

pointDocuments.add(createPointDocument(point));

}

**return** pointDocuments;

}

}

## **MoveFiguresController**

**package** com.figures.figuresfx;

**import** **com.figures.exceptions.NullInputException**;

**import** **com.figures.storage.Storage**;

**import** **javafx.collections.FXCollections**;

**import** **javafx.event.ActionEvent**;

**import** **javafx.fxml.FXML**;

**import** **javafx.scene.Node**;

**import** **javafx.scene.canvas.GraphicsContext**;

**import** **javafx.scene.control.\***;

**import** **javafx.scene.layout.Pane**;

**import** **javafx.scene.paint.Color**;

**import** **javafx.stage.Stage**;

**import** **lombok.extern.java.Log**;

**import** **org.figures.figures.points.Point2D**;

**import** **java.util.Objects**;

**@Log**

**public** **class** **MoveFiguresController** {

**@FXML**

**private** Spinner<String> domainUpDown1;

**@FXML**

**private** Pane panel1;

**@FXML**

**public** ComboBox<String> comboBox2;

**@FXML**

**public** ComboBox<String> comboBox3;

**@FXML**

**private** TextField textBox38;

**@FXML**

**private** TextField textBox39;

**@FXML**

**private** TextField textBox40;

**@FXML**

**private** Pane pn111;

**@FXML**

**private** Pane pn222;

**@FXML**

**private** Pane pn333;

**private** MainController mainController;

**private** AddFiguresController addFiguresController;

**private** CrossFiguresController crossFiguresController;

**private** DeleteFiguresController deleteFiguresController;

**public** **void** **setControllers**(MainController mainController, AddFiguresController addFiguresController, CrossFiguresController crossFiguresController, DeleteFiguresController deleteFiguresController) {

**this**.mainController = mainController;

**this**.addFiguresController = addFiguresController;

**this**.crossFiguresController = crossFiguresController;

**this**.deleteFiguresController = deleteFiguresController;

}

**private** FiguresApp application;

**public** **void** **setApplication**(FiguresApp application) {

**this**.application = application;

}

**public** **void** **initialize**(){

comboBox2.getItems().addAll("Сдвиг",

"Поворот",

"Симметрия"

);

String[] items = {"x", "y"};

SpinnerValueFactory<String> valueFactory\_dup = **new** SpinnerValueFactory.ListSpinnerValueFactory<>(FXCollections.observableArrayList(items));

domainUpDown1.setValueFactory(valueFactory\_dup);

**for**(**int** i = **0**; i < Storage.getListFigures().size(); i++) {

String figure = Storage.nameFigures.get(i);

**switch** (figure) {

**case** "Окружность":

comboBox3.getItems().add("Окружность: " + "Circle" + Storage.getListFigures().get(i).toString());

//deleteFiguresController.comboBox4.getItems().add("Окружность: " + "Circle" + Storage.getListFigures().get(Storage.getFiguresCount()).toString());

**break**;

**case** "Отрезок":

comboBox3.getItems().add("Отрезок: " + "Segment" + Storage.getListFigures().get(i).toString());

//deleteFiguresController.comboBox4.getItems().add("Отрезок: " + "Segment" + Storage.getListFigures().get(Storage.getFiguresCount()).toString());

**break**;

**case** "Ломанная": // Если фигура - полилиния

**case** "Многоугольник": // Если фигура - правильный n-угольник

**case** "Четырёхугольник": // Если фигура - квадрат

**case** "Треугольник": // Если фигура - равнобедренный треугольник

**case** "Трапеция": // Если фигура - трапеция

**case** "Прямоугольник": // Если фигура - прямоугольник

// Добавление фигуры в список в зависимости от типа

**if** (figure == "Ломанная") {

comboBox3.getItems().add("Ломанная: " + "Polyline" + Storage.getListFigures().get(i).toString());

//deleteFiguresController.comboBox4.getItems().add("Ломанная: " + "Polyline" + Storage.getListFigures().get(Storage.getFiguresCount()).toString());

}

**if** (figure == "Многоугольник") {

comboBox3.getItems().add("Многоугольник: " + "NGon" + Storage.getListFigures().get(i).toString());

//deleteFiguresController.comboBox4.getItems().add("Многоугольник: " + "NGon" + Storage.getListFigures().get(Storage.getFiguresCount()).toString());

}

**if** (figure == "Четырёхугольник") {

comboBox3.getItems().add("Четырёхугольник: " + "QGon" + Storage.getListFigures().get(i).toString());

//deleteFiguresController.comboBox4.getItems().add("Четырёхугольник: " + "QGon" + Storage.getListFigures().get(Storage.getFiguresCount()).toString());

}

**if** (figure == "Треугольник") {

comboBox3.getItems().add("Треугольник: " + "TGon" + Storage.getListFigures().get(i).toString());

//deleteFiguresController.comboBox4.getItems().add("Треугольник: " + "TGon" + Storage.getListFigures().get(Storage.getFiguresCount()).toString());

}

**if** (figure == "Трапеция") {

comboBox3.getItems().add("Трапеция: " + "Trapeze" + Storage.getListFigures().get(i).toString());

//deleteFiguresController.comboBox4.getItems().add("Трапеция: " + "Trapeze" + Storage.getListFigures().get(Storage.getFiguresCount()).toString());

}

**if** (figure == "Прямоугольник") {

comboBox3.getItems().add("Прямоугольник: " + "Rectangle" + Storage.getListFigures().get(i).toString());

//deleteFiguresController.comboBox4.getItems().add("Прямоугольник: " + "Rectangle" + Storage.getListFigures().get(Storage.getFiguresCount()).toString());

}

**break**;

}

}

}

**@FXML**

**private** **void** **comboBox2\_SelectedIndexChanged**(ActionEvent actionEvent) {

**int** selectedIndex = comboBox2.getSelectionModel().getSelectedIndex();

**switch** (selectedIndex) {

**case** **0**:

pn111.setDisable(**false**);

pn222.setDisable(**true**);

pn333.setDisable(**true**);

**break**;

**case** **1**:

pn111.setDisable(**true**);

pn222.setDisable(**false**);

pn333.setDisable(**true**);

**break**;

**case** **2**:

pn111.setDisable(**true**);

pn222.setDisable(**true**);

pn333.setDisable(**false**);

**break**;

}

}

**@FXML**

**private** **void** **button5\_Click**(ActionEvent actionEvent) **throws** NullInputException {

**if**(Storage.getFiguresCount() == **0**){

deleteFiguresController.comboBox4.getSelectionModel().clearSelection();

}

**else** **if**(deleteFiguresController.comboBox4.getSelectionModel().getSelectedIndex() < **0**){

deleteFiguresController.comboBox4.getSelectionModel().select(**0**);

}

**try** {

**if** (comboBox3.getSelectionModel().getSelectedIndex() == -**1** || comboBox2.getSelectionModel().getSelectedIndex() == -**1** || (comboBox2.getSelectionModel().getSelectedIndex() == -**1** && Storage.getFiguresCount() == **0**)) {

**throw** **new** **NullInputException**();

} **else** {

**if** (comboBox2.getSelectionModel().getSelectedIndex() == **0** && (textBox38.getText().isEmpty() || textBox39.getText().isEmpty())) {

**throw** **new** **NullInputException**();

}

**if** (comboBox2.getSelectionModel().getSelectedIndex() == **1** && textBox40.getText().isEmpty()) {

**throw** **new** **NullInputException**();

}

**if** (comboBox2.getSelectionModel().getSelectedIndex() == **2** && domainUpDown1.getValue() == **null**) {

**throw** **new** **NullInputException**();

}

}

String move = comboBox2.getSelectionModel().getSelectedItem();

String type;

String originalString;

String newString;

**switch** (move) {

**case** "Поворот":

**if** (comboBox3.getSelectionModel().getSelectedIndex() < **0**) {

comboBox3.getSelectionModel().select(**0**);

}

**if** (deleteFiguresController.comboBox4.getSelectionModel().getSelectedIndex() < **0**) {

deleteFiguresController.comboBox4.getSelectionModel().select(**0**);

}

**if** (crossFiguresController.comboBox5.getSelectionModel().getSelectedIndex() < **0**) {

crossFiguresController.comboBox5.getSelectionModel().select(**0**);

}

**if** (crossFiguresController.comboBox6.getSelectionModel().getSelectedIndex() < **0**) {

crossFiguresController.comboBox6.getSelectionModel().select(**0**);

}

log.info(Storage.nameFigures.get(comboBox3.getSelectionModel().getSelectedIndex()) + Storage.getListFigures().get(comboBox3.getSelectionModel().getSelectedIndex()) + " был(а) повернута");

Storage.getListFigures().add(Storage.getListFigures().get(comboBox3.getSelectionModel().getSelectedIndex()).rot(Double.parseDouble(textBox40.getText())));

Storage.getListFigures().remove(comboBox3.getSelectionModel().getSelectedIndex());

comboBox3.getItems().remove(comboBox3.getSelectionModel().getSelectedIndex());

type = Storage.getListFigures().get(Storage.getListFigures().size() - **1**).getClass().getSimpleName();

originalString = Storage.getListFigures().get(Storage.getListFigures().size() - **1**).toString();

newString = originalString.replace("FiguresFX.", "");

type = type.replace("FiguresFX.", "");

**if** ("Circle".equals(type)) {

comboBox3.getItems().add("Окружность: " + type + newString);

log.info("Новая Окружность: " + type + newString);

} **else** **if** ("NGon".equals(type)) {

comboBox3.getItems().add("Многоугольник: " + type + newString);

log.info("Новый Многоугольник: " + type + newString);

} **else** **if** ("Trapeze".equals(type)) {

comboBox3.getItems().add("Трапеция: " + type + newString);

log.info("Новая Трапеция: " + type + newString);

} **else** **if** ("Rectangle".equals(type)) {

comboBox3.getItems().add("Прямоугольник: " + type + newString);

log.info("Новый Прямоугольник: " + type + newString);

} **else** **if** ("TGon".equals(type)) {

comboBox3.getItems().add("Треугольник: " + type + newString);

log.info("Новый Треугольник: " + type + newString);

} **else** **if** ("Polyline".equals(type)) {

comboBox3.getItems().add("Ломанная: " + type + newString);

log.info("Новая Ломанная: " + type + newString);

} **else** **if** ("QGon".equals(type)) {

comboBox3.getItems().add("Четырёхугольник: " + type + newString);

log.info("Новый Четырёхугольник: " + type + newString);

} **else** **if** ("Segment".equals(type)) {

comboBox3.getItems().add("Отрезок: " + type + newString);

log.info("Новый Отрезок: " + type + newString);

}

**break**;

**case** "Симметрия":

String value = domainUpDown1.getValue();

**if** (Objects.equals(value, "x")) {

**if** (comboBox3.getSelectionModel().getSelectedIndex() < **0**) {

comboBox3.getSelectionModel().select(**0**);

}

**if** (deleteFiguresController.comboBox4.getSelectionModel().getSelectedIndex() < **0**) {

deleteFiguresController.comboBox4.getSelectionModel().select(**0**);

}

**if** (crossFiguresController.comboBox5.getSelectionModel().getSelectedIndex() < **0**) {

crossFiguresController.comboBox5.getSelectionModel().select(**0**);

}

**if** (crossFiguresController.comboBox6.getSelectionModel().getSelectedIndex() < **0**) {

crossFiguresController.comboBox6.getSelectionModel().select(**0**);

}

log.info(Storage.nameFigures.get(comboBox3.getSelectionModel().getSelectedIndex()) + Storage.getListFigures().get(comboBox3.getSelectionModel().getSelectedIndex()) + " был(а) симметрично отражена");

Storage.getListFigures().add(Storage.getListFigures().get(comboBox3.getSelectionModel().getSelectedIndex()).symAxis(**0**));

Storage.getListFigures().remove(comboBox3.getSelectionModel().getSelectedIndex());

comboBox3.getItems().remove(comboBox3.getSelectionModel().getSelectedIndex());

type = Storage.getListFigures().get(Storage.getListFigures().size() - **1**).getClass().getSimpleName();

originalString = Storage.getListFigures().get(Storage.getListFigures().size() - **1**).toString();

newString = originalString.replace("FiguresFX.", "");

type = type.replace("FiguresFX.", "");

**if** ("Circle".equals(type)) {

comboBox3.getItems().add("Окружность: " + type + newString);

log.info("Новая Окружность: " + type + newString);

} **else** **if** ("NGon".equals(type)) {

comboBox3.getItems().add("Многоугольник: " + type + newString);

log.info("Новый Многоугольник: " + type + newString);

} **else** **if** ("Trapeze".equals(type)) {

comboBox3.getItems().add("Трапеция: " + type + newString);

log.info("Новая Трапеция: " + type + newString);

} **else** **if** ("Rectangle".equals(type)) {

comboBox3.getItems().add("Прямоугольник: " + type + newString);

log.info("Новый Прямоугольник: " + type + newString);

} **else** **if** ("TGon".equals(type)) {

comboBox3.getItems().add("Треугольник: " + type + newString);

log.info("Новый Треугольник: " + type + newString);

} **else** **if** ("Polyline".equals(type)) {

comboBox3.getItems().add("Ломанная: " + type + newString);

log.info("Новая Ломанная: " + type + newString);

} **else** **if** ("QGon".equals(type)) {

comboBox3.getItems().add("Четырёхугольник: " + type + newString);

log.info("Новый Четырёхугольник: " + type + newString);

} **else** **if** ("Segment".equals(type)) {

comboBox3.getItems().add("Отрезок: " + type + newString);

log.info("Новый Отрезок: " + type + newString);

}

} **else** **if** (Objects.equals(value, "y")) {

**if** (comboBox3.getSelectionModel().getSelectedIndex() < **0**) {

comboBox3.getSelectionModel().select(**0**);

}

**if** (deleteFiguresController.comboBox4.getSelectionModel().getSelectedIndex() < **0**) {

deleteFiguresController.comboBox4.getSelectionModel().select(**0**);

}

**if** (crossFiguresController.comboBox5.getSelectionModel().getSelectedIndex() < **0**) {

crossFiguresController.comboBox5.getSelectionModel().select(**0**);

}

**if** (crossFiguresController.comboBox6.getSelectionModel().getSelectedIndex() < **0**) {

crossFiguresController.comboBox6.getSelectionModel().select(**0**);

}

log.info(Storage.nameFigures.get(comboBox3.getSelectionModel().getSelectedIndex()) + Storage.getListFigures().get(comboBox3.getSelectionModel().getSelectedIndex()) + " был(а) симметрично отражена");

Storage.getListFigures().add(Storage.getListFigures().get(comboBox3.getSelectionModel().getSelectedIndex()).symAxis(**1**));

Storage.getListFigures().remove(comboBox3.getSelectionModel().getSelectedIndex());

comboBox3.getItems().remove(comboBox3.getSelectionModel().getSelectedIndex());

type = Storage.getListFigures().get(Storage.getListFigures().size() - **1**).getClass().getSimpleName();

originalString = Storage.getListFigures().get(Storage.getListFigures().size() - **1**).toString();

newString = originalString.replace("FiguresFX.", "");

type = type.replace("FiguresFX.", "");

**if** ("Circle".equals(type)) {

comboBox3.getItems().add("Окружность: " + type + newString);

log.info("Новая Окружность" + type + newString);

} **else** **if** ("NGon".equals(type)) {

comboBox3.getItems().add("Многоугольник: " + type + newString);

log.info("Новый Многоугольник" + type + newString);

} **else** **if** ("Trapeze".equals(type)) {

comboBox3.getItems().add("Трапеция: " + type + newString);

log.info("Новая Трапеция" + type + newString);

} **else** **if** ("Rectangle".equals(type)) {

comboBox3.getItems().add("Прямоугольник: " + type + newString);

log.info("Новый Прямоугольник" + type + newString);

} **else** **if** ("TGon".equals(type)) {

comboBox3.getItems().add("Треугольник: " + type + newString);

log.info("Новый Треугольник" + type + newString);

} **else** **if** ("Polyline".equals(type)) {

comboBox3.getItems().add("Ломанная: " + type + newString);

log.info("Новая Ломанная" + type + newString);

} **else** **if** ("QGon".equals(type)) {

comboBox3.getItems().add("Четырёхугольник: " + type + newString);

log.info("Новый Четырёхугольник" + type + newString);

} **else** **if** ("Segment".equals(type)) {

comboBox3.getItems().add("Отрезок: " + type + newString);

log.info("Новый Отрезок" + type + newString);

}

} **else** **if** (Objects.equals(value, "z")) {

**if** (comboBox3.getSelectionModel().getSelectedIndex() < **0**) {

comboBox3.getSelectionModel().select(**0**);

}

**if** (deleteFiguresController.comboBox4.getSelectionModel().getSelectedIndex() < **0**) {

deleteFiguresController.comboBox4.getSelectionModel().select(**0**);

}

**if** (crossFiguresController.comboBox5.getSelectionModel().getSelectedIndex() < **0**) {

crossFiguresController.comboBox5.getSelectionModel().select(**0**);

}

**if** (crossFiguresController.comboBox6.getSelectionModel().getSelectedIndex() < **0**) {

crossFiguresController.comboBox6.getSelectionModel().select(**0**);

}

log.info(Storage.nameFigures.get(comboBox3.getSelectionModel().getSelectedIndex()) + Storage.getListFigures().get(comboBox3.getSelectionModel().getSelectedIndex()) + " был(а) симметрично отражена");

Storage.getListFigures().add(Storage.getListFigures().get(comboBox3.getSelectionModel().getSelectedIndex()).symAxis(**2**));

Storage.getListFigures().remove(comboBox3.getSelectionModel().getSelectedIndex());

comboBox3.getItems().remove(comboBox3.getSelectionModel().getSelectedIndex());

type = Storage.getListFigures().get(Storage.getListFigures().size() - **1**).getClass().getSimpleName();

originalString = Storage.getListFigures().get(Storage.getListFigures().size() - **1**).toString();

newString = originalString.replace("FiguresFX.", "");

type = type.replace("FiguresFX.", "");

**if** ("Circle".equals(type)) {

comboBox3.getItems().add("Окружность: " + type + newString);

log.info("Новая Окружность: " + type + newString);

} **else** **if** ("NGon".equals(type)) {

comboBox3.getItems().add("Многоугольник: " + type + newString);

log.info("Новый Многоугольник: " + type + newString);

} **else** **if** ("Trapeze".equals(type)) {

comboBox3.getItems().add("Трапеция: " + type + newString);

log.info("Новая Трапеция: " + type + newString);

} **else** **if** ("Rectangle".equals(type)) {

comboBox3.getItems().add("Прямоугольник: " + type + newString);

log.info("Новая Прямоугольник: " + type + newString);

} **else** **if** ("TGon".equals(type)) {

comboBox3.getItems().add("Треугольник: " + type + newString);

log.info("Новая Треугольник: " + type + newString);

} **else** **if** ("Polyline".equals(type)) {

comboBox3.getItems().add("Ломанная: " + type + newString);

log.info("Новая Ломанная: " + type + newString);

} **else** **if** ("QGon".equals(type)) {

comboBox3.getItems().add("Четырёхугольник: " + type + newString);

log.info("Новая Четырёхугольник: " + type + newString);

} **else** **if** ("Segment".equals(type)) {

comboBox3.getItems().add("Отрезок: " + type + newString);

log.info("Новый Отрезок: " + type + newString);

}

}

**break**;

**case** "Сдвиг":

**if** (comboBox3.getSelectionModel().getSelectedIndex() < **0**) {

comboBox3.getSelectionModel().select(**0**);

}

**if** (deleteFiguresController.comboBox4.getSelectionModel().getSelectedIndex() < **0**) {

deleteFiguresController.comboBox4.getSelectionModel().select(**0**);

}

**if** (crossFiguresController.comboBox5.getSelectionModel().getSelectedIndex() < **0**) {

crossFiguresController.comboBox5.getSelectionModel().select(**0**);

}

**if** (crossFiguresController.comboBox6.getSelectionModel().getSelectedIndex() < **0**) {

crossFiguresController.comboBox6.getSelectionModel().select(**0**);

}

log.info(Storage.nameFigures.get(comboBox3.getSelectionModel().getSelectedIndex()) + Storage.getListFigures().get(comboBox3.getSelectionModel().getSelectedIndex()) + " был(а) сдвинута");

Storage.getListFigures().add(Storage.getListFigures().get(comboBox3.getSelectionModel().getSelectedIndex()).shift(**new** Point2D(**new** **double**[]{Double.parseDouble(textBox38.getText()), Double.parseDouble(textBox39.getText())})));

Storage.getListFigures().remove(comboBox3.getSelectionModel().getSelectedIndex());

comboBox3.getItems().remove(comboBox3.getSelectionModel().getSelectedIndex());

type = Storage.getListFigures().get(Storage.getListFigures().size() - **1**).getClass().getSimpleName();

originalString = Storage.getListFigures().get(Storage.getListFigures().size() - **1**).toString();

newString = originalString.replace("FiguresFX.", "");

type = type.replace("FiguresFX.", "");

**if** ("Circle".equals(type)) {

comboBox3.getItems().add("Окружность: " + type + newString);

log.info("Новая Окружность:" + type + newString);

} **else** **if** ("NGon".equals(type)) {

comboBox3.getItems().add("Многоугольник: " + type + newString);

log.info("Новый Многоугольник:" + type + newString);

} **else** **if** ("Trapeze".equals(type)) {

comboBox3.getItems().add("Трапеция: " + type + newString);

log.info("Новая Трапеция:" + type + newString);

} **else** **if** ("Rectangle".equals(type)) {

comboBox3.getItems().add("Прямоугольник: " + type + newString);

log.info("Новый Прямоугольник:" + type + newString);

} **else** **if** ("TGon".equals(type)) {

comboBox3.getItems().add("Треугольник: " + type + newString);

log.info("Новый Треугольник:" + type + newString);

} **else** **if** ("Polyline".equals(type)) {

comboBox3.getItems().add("Ломанная: " + type + newString);

log.info("Новая Ломанная:" + type + newString);

} **else** **if** ("QGon".equals(type)) {

comboBox3.getItems().add("Четырёхугольник: " + type + newString);

log.info("Новый Четырёхугольник:" + type + newString);

} **else** **if** ("Segment".equals(type)) {

comboBox3.getItems().add("Отрезок: " + type + newString);

log.info("Новый Отрезок:" + type + newString);

}

**break**;

}

Storage.nameFigures.clear();

**for**(**int** i = **0**; i < Storage.figuresCount; i++ )

{

type = Storage.getListFigures().get(i).getClass().getSimpleName();

type = type.replace("FiguresFX.", "");

**if** ("Circle".equals(type)) {

Storage.nameFigures.add("Окружность");

} **else** **if** ("NGon".equals(type)) {

Storage.nameFigures.add("Многоугольник");

} **else** **if** ("Trapeze".equals(type)) {

Storage.nameFigures.add("Трапеция");

} **else** **if** ("Rectangle".equals(type)) {

Storage.nameFigures.add("Прямоугольник");

} **else** **if** ("TGon".equals(type)) {

Storage.nameFigures.add("Треугольник");

} **else** **if** ("Polyline".equals(type)) {

Storage.nameFigures.add("Ломанная");

} **else** **if** ("QGon".equals(type)) {

Storage.nameFigures.add("Четырёхугольник");

} **else** **if** ("Segment".equals(type)) {

Storage.nameFigures.add("Отрезок");

}

}

**if** (mainController.canvas != **null**) {

// Очищаем содержимое холста (canvas) с белым цветом

GraphicsContext gc = mainController.canvas.getGraphicsContext2D();

gc.setFill(Color.WHITE);

gc.clearRect(**0**, **0**, mainController.canvas.getWidth(), mainController.canvas.getHeight());

// Вызываем метод requestFocus(), чтобы перерисовать холст

mainController.canvas.requestFocus();

}

**assert** mainController.canvas != **null**;

application.drawAxes(mainController.canvas.getWidth(), mainController.canvas.getHeight());

**for**(**int** i = **0**; i < Storage.getListFigures().size(); i++)

{

mainController.draw(Storage.getListFigures().get(i), Color.BLACK);

}

Alert alert = **new** Alert(Alert.AlertType.INFORMATION);

alert.setTitle("Успех");

alert.setHeaderText(**null**);

alert.setContentText("Фигура успешно перемещена");

alert.showAndWait();

}

**catch** (NullInputException ex) {

Alert alert = **new** Alert(Alert.AlertType.WARNING);

alert.setTitle("Ошибка");

alert.setHeaderText("NullInputException");

alert.setContentText(ex.toString());

alert.showAndWait();

} **catch** (IllegalArgumentException ex) {

Alert alert = **new** Alert(Alert.AlertType.WARNING);

alert.setTitle("Ошибка");

alert.setHeaderText("IllegalArgumentException ex");

alert.setContentText(ex.getMessage());

alert.showAndWait();

}

}

**public** **void** **CancelButton\_Click**(ActionEvent actionEvent) {

Stage stage = (Stage) ((Node) actionEvent.getSource()).getScene().getWindow();

// Закрыть текущую сцену (форму)

stage.close();

}

}

## **PerimeterController**

**package** com.figures.figuresfx;

**import** **com.figures.storage.Storage**;

**import** **javafx.event.ActionEvent**;

**import** **javafx.fxml.FXML**;

**import** **javafx.scene.Node**;

**import** **javafx.scene.canvas.Canvas**;

**import** **javafx.scene.canvas.GraphicsContext**;

**import** **javafx.scene.control.Alert**;

**import** **javafx.scene.control.ComboBox**;

**import** **javafx.scene.paint.Color**;

**import** **javafx.stage.Stage**;

**import** **lombok.extern.java.Log**;

**@Log**

**public** **class** **PerimeterFigureController** {

**@FXML**

**public** ComboBox<String> comboBoxp;

**private** MainController mainController;

**public** **void** **setControllers**(MainController mainController) {

**this**.mainController = mainController;

}

**private** FiguresApp application;

**public** **void** **setApplication**(FiguresApp application) {

**this**.application = application;

}

**public** **void** **initialize**(){

**for**(**int** i = **0**; i < Storage.getListFigures().size(); i++) {

String figure = Storage.nameFigures.get(i);

**switch** (figure) {

**case** "Окружность":

comboBoxp.getItems().add("Окружность: " + "Circle" + Storage.getListFigures().get(i).toString());

**break**;

**case** "Отрезок":

comboBoxp.getItems().add("Отрезок: " + "Segment" + Storage.getListFigures().get(i).toString());

**break**;

**case** "Ломанная": // Если фигура - полилиния

**case** "Многоугольник": // Если фигура - правильный n-угольник

**case** "Четырёхугольник": // Если фигура - квадрат

**case** "Треугольник": // Если фигура - равнобедренный треугольник

**case** "Трапеция": // Если фигура - трапеция

**case** "Прямоугольник": // Если фигура - прямоугольник

// Добавление фигуры в список в зависимости от типа

**if** (figure == "Ломанная") {

comboBoxp.getItems().add("Ломанная: " + "Polyline" + Storage.getListFigures().get(i).toString());

}

**if** (figure == "Многоугольник") {

comboBoxp.getItems().add("Многоугольник: " + "NGon" + Storage.getListFigures().get(i).toString());

}

**if** (figure == "Четырёхугольник") {

comboBoxp.getItems().add("Четырёхугольник: " + "QGon" + Storage.getListFigures().get(i).toString());

}

**if** (figure == "Треугольник") {

comboBoxp.getItems().add("Треугольник: " + "TGon" + Storage.getListFigures().get(i).toString());

}

**if** (figure == "Трапеция") {

comboBoxp.getItems().add("Трапеция: " + "Trapeze" + Storage.getListFigures().get(i).toString());

}

**if** (figure == "Прямоугольник") {

comboBoxp.getItems().add("Прямоугольник: " + "Rectangle" + Storage.getListFigures().get(i).toString());

}

**break**;

}

String type;

Storage.nameFigures.clear();

**for**(**int** j = **0**; j < Storage.figuresCount; j++ )

{

type = Storage.getListFigures().get(j).getClass().getSimpleName();

type = type.replace("FiguresFX.", "");

**if** ("Circle".equals(type)) {

Storage.nameFigures.add("Окружность");

} **else** **if** ("NGon".equals(type)) {

Storage.nameFigures.add("Многоугольник");

} **else** **if** ("Trapeze".equals(type)) {

Storage.nameFigures.add("Трапеция");

} **else** **if** ("Rectangle".equals(type)) {

Storage.nameFigures.add("Прямоугольник");

} **else** **if** ("TGon".equals(type)) {

Storage.nameFigures.add("Треугольник");

} **else** **if** ("Polyline".equals(type)) {

Storage.nameFigures.add("Ломанная");

} **else** **if** ("QGon".equals(type)) {

Storage.nameFigures.add("Четырёхугольник");

} **else** **if** ("Segment".equals(type)) {

Storage.nameFigures.add("Отрезок");

}

}

}

}

//Периметр

**@FXML**

**public** **void** **buttons\_Click**(ActionEvent actionEvent) {

**try**

{

Canvas canvas = mainController.canvas;

GraphicsContext gc = mainController.canvas.getGraphicsContext2D();

gc.clearRect(**0**, **0**, canvas.getWidth(), canvas.getHeight());

application.drawAxes(canvas.getWidth(), canvas.getHeight());

**for**(**int** i = **0**; i < Storage.figuresCount; i++){

mainController.draw(Storage.getListFigures().get(i), Color.BLACK);

}

**double** perimetr = Storage.listFigures.get(comboBoxp.getSelectionModel().getSelectedIndex()).length();

log.info("Периметр фигуры " + Storage.nameFigures.get(comboBoxp.getSelectionModel().getSelectedIndex()) + ":" + perimetr);

mainController.label61.setText("Периметр: ");

mainController.textBox41.setText(String.valueOf(perimetr));

Alert alert = **new** Alert(Alert.AlertType.INFORMATION);

alert.setTitle("Успех");

alert.setHeaderText(**null**);

alert.setContentText("Периметр подсчитан");

alert.showAndWait();

}

**catch**(IllegalArgumentException ex)

{

Alert alert = **new** Alert(Alert.AlertType.WARNING);

alert.setTitle("Ошибка");

alert.setHeaderText("IllegalArgumentException ex");

alert.setContentText(ex.getMessage());

alert.showAndWait();

}

}

**public** **void** **CancelButton\_Click**(ActionEvent actionEvent) {

Stage stage = (Stage) ((Node) actionEvent.getSource()).getScene().getWindow();

// Закрыть текущую сцену (форму)

stage.close();

}

}

## **SquareFigureController**

**package** com.figures.figuresfx;

**import** **com.figures.storage.Storage**;

**import** **javafx.event.ActionEvent**;

**import** **javafx.fxml.FXML**;

**import** **javafx.scene.Node**;

**import** **javafx.scene.canvas.Canvas**;

**import** **javafx.scene.canvas.GraphicsContext**;

**import** **javafx.scene.control.Alert**;

**import** **javafx.scene.control.ComboBox**;

**import** **javafx.scene.paint.Color**;

**import** **javafx.stage.Stage**;

**import** **lombok.extern.java.Log**;

**@Log**

**public** **class** **SquareFigureController** {

**@FXML**

**public** ComboBox<String> comboBoxs;

**private** MainController mainController;

**public** **void** **setControllers**(MainController mainController) {

**this**.mainController = mainController;

}

**private** FiguresApp application;

**public** **void** **setApplication**(FiguresApp application) {

**this**.application = application;

}

**public** **void** **initialize**(){

**for**(**int** i = **0**; i < Storage.getListFigures().size(); i++) {

String figure = Storage.nameFigures.get(i);

**switch** (figure) {

**case** "Окружность":

comboBoxs.getItems().add("Окружность: " + "Circle" + Storage.getListFigures().get(i).toString());

**break**;

**case** "Отрезок":

comboBoxs.getItems().add("Отрезок: " + "Segment" + Storage.getListFigures().get(i).toString());

**break**;

**case** "Ломанная": // Если фигура - полилиния

**case** "Многоугольник": // Если фигура - правильный n-угольник

**case** "Четырёхугольник": // Если фигура - квадрат

**case** "Треугольник": // Если фигура - равнобедренный треугольник

**case** "Трапеция": // Если фигура - трапеция

**case** "Прямоугольник": // Если фигура - прямоугольник

// Добавление фигуры в список в зависимости от типа

**if** (figure == "Ломанная") {

comboBoxs.getItems().add("Ломанная: " + "Polyline" + Storage.getListFigures().get(i).toString());

}

**if** (figure == "Многоугольник") {

comboBoxs.getItems().add("Многоугольник: " + "NGon" + Storage.getListFigures().get(i).toString());

}

**if** (figure == "Четырёхугольник") {

comboBoxs.getItems().add("Четырёхугольник: " + "QGon" + Storage.getListFigures().get(i).toString());

}

**if** (figure == "Треугольник") {

comboBoxs.getItems().add("Треугольник: " + "TGon" + Storage.getListFigures().get(i).toString());

}

**if** (figure == "Трапеция") {

comboBoxs.getItems().add("Трапеция: " + "Trapeze" + Storage.getListFigures().get(i).toString());

}

**if** (figure == "Прямоугольник") {

comboBoxs.getItems().add("Прямоугольник: " + "Rectangle" + Storage.getListFigures().get(i).toString());

}

**break**;

}

}

String type;

Storage.nameFigures.clear();

**for**(**int** j = **0**; j < Storage.figuresCount; j++ )

{

type = Storage.getListFigures().get(j).getClass().getSimpleName();

type = type.replace("FiguresFX.", "");

**if** ("Circle".equals(type)) {

Storage.nameFigures.add("Окружность");

} **else** **if** ("NGon".equals(type)) {

Storage.nameFigures.add("Многоугольник");

} **else** **if** ("Trapeze".equals(type)) {

Storage.nameFigures.add("Трапеция");

} **else** **if** ("Rectangle".equals(type)) {

Storage.nameFigures.add("Прямоугольник");

} **else** **if** ("TGon".equals(type)) {

Storage.nameFigures.add("Треугольник");

} **else** **if** ("Polyline".equals(type)) {

Storage.nameFigures.add("Ломанная");

} **else** **if** ("QGon".equals(type)) {

Storage.nameFigures.add("Четырёхугольник");

} **else** **if** ("Segment".equals(type)) {

Storage.nameFigures.add("Отрезок");

}

}

}

**@FXML**

**public** **void** **buttons\_Click**(ActionEvent actionEvent) {

**try**

{

Canvas canvas = mainController.canvas;

GraphicsContext gc = mainController.canvas.getGraphicsContext2D();

gc.clearRect(**0**, **0**, canvas.getWidth(), canvas.getHeight());

application.drawAxes(canvas.getWidth(), canvas.getHeight());

**for**(**int** i = **0**; i < Storage.figuresCount; i++){

mainController.draw(Storage.getListFigures().get(i), Color.BLACK);

}

**double** square = Storage.listFigures.get(comboBoxs.getSelectionModel().getSelectedIndex()).square();

log.info("Площадь фигуры " + Storage.nameFigures.get(comboBoxs.getSelectionModel().getSelectedIndex()) + ":" + square);

mainController.label61.setText("Площадь: ");

mainController.textBox41.setText(String.valueOf(square));

Alert alert = **new** Alert(Alert.AlertType.INFORMATION);

alert.setTitle("Успех");

alert.setHeaderText(**null**);

alert.setContentText("Площадь подсчитана");

alert.showAndWait();

}

**catch**(IllegalArgumentException ex)

{

Alert alert = **new** Alert(Alert.AlertType.WARNING);

alert.setTitle("Ошибка");

alert.setHeaderText("IllegalArgumentException ex");

alert.setContentText(ex.getMessage());

alert.showAndWait();

}

}

**public** **void** **CancelButton\_Click**(ActionEvent actionEvent) {

Stage stage = (Stage) ((Node) actionEvent.getSource()).getScene().getWindow();

// Закрыть текущую сцену (форму)

stage.close();

}

}

## **Storage**

**package** com.figures.storage;

**import** **org.figures.interfaces.IShape**;

**import** **java.util.ArrayList**;

**import** **java.util.List**;

**public** **class** **Storage** {

**public** **static** **int** figuresCount = **0**;

**public** **static** **int** pointsCount = **0**;

**public** **static** List<IShape> listFigures = **new** ArrayList<>();

**public** **static** List<Double> listPoints = **new** ArrayList<>();

**public** **static** List<String> nameFigures = **new** ArrayList<>();

**public** **static** **void** **setPointsCount**(**int** value) {

pointsCount = value;

}

**public** **void** **setFiguresCount**(**int** value) {

figuresCount = value;

}

**public** **void** **incPointsCount**(){

pointsCount++;

}

**public** **static** **void** **incFiguresCount**(){

figuresCount++;

}

**public** **static** **void** **decFiguresCount**(){

figuresCount--;

}

**public** **static** **int** **getPointsCount**() {

**return** pointsCount;

}

**public** **static** **int** **getFiguresCount**() {

**return** figuresCount;

}

**public** **static** List<Double> **getListPoints**(){

**return** listPoints;

}

**public** **static** List<IShape> **getListFigures**(){

**return** listFigures;

}

}