Отчет

Малахаева Валерия

09-132

2023

## Консольное приложение

1. OpenFigure

Этот абстрактный класс служит основой для создания конкретных геометрических фигур, предоставляя общий интерфейс для расчета длины, смещения, вращения, симметрии и определения пересечений. Подклассы должны реализовать абстрактные методы в соответствии с конкретным типом геометрической фигуры.

Абстрактные методы:

* double length(): метод, возвращающий длину фигуры.
* IShape shift(Point2D a): метод, используемый для смещения фигуры на указанный вектор a.
* IShape rot(double phi): метод, используемый для поворота фигуры на указанный угол phi.
* IShape symAxis(int i): метод, используемый для выполнения симметрии относительно оси с индексом i.
* boolean cross(IShape i): метод, возвращающий булево значение, указывающее, пересекается ли данная фигура с другой формой i.

Реализация метода square():

* Метод square() возвращает 0. Вероятно, предназначен для переопределения в подклассах.

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);  
}

1. Point

Класс Point в пакете org.figures.figures.points предоставляет представление точки в n-мерном пространстве. Вот основные характеристики этого класса:

1. Поля класса:
   * dim: Размерность пространства (защищенное поле с геттером).
   * x: Массив координат точки (защищенное поле с геттером).
2. Конструкторы:
   * public Point(int dim): Конструктор с одним параметром, который инициализирует размерность пространства и создает массив координат.
   * public Point(int dim, double[] x): Конструктор с двумя параметрами, который инициализирует размерность пространства и массив координат, проверяя их соответствие.
3. Методы доступа к координатам:
   * public double getX(int i): Получение значения i-й координаты точки.
   * public void setX(double[] x): Установка нового массива координат точки, проверяя соответствие размерности.
   * public void setX(double x, int i): Установка нового значения i-й координаты точки, проверяя правильность индекса.
4. Методы для операций над точками:
   * public double abs(): Вычисление модуля точки (расстояния от начала координат).
   * public static Point add(Point a, Point b): Сложение двух точек (статический метод).
   * public Point add(Point b): Сложение текущей точки с другой точкой.
   * public static Point sub(Point a, Point b): Вычитание одной точки из другой (статический метод).
   * public Point sub(Point b): Вычитание из текущей точки другой точки.
   * public static Point mult(Point a, double r): Умножение точки на число (статический метод).
   * public Point mult(double r): Умножение текущей точки на число.
   * public static double mult(Point a, Point b): Вычисление скалярного произведения двух точек (статический метод).
   * public double mult(Point b): Вычисление скалярного произведения текущей точки с другой точкой.
   * public static Point symAxis(Point a, int i): Симметричное отражение точки относительно i-ой оси (статический метод).
   * public Point symAxis(int i): Симметричное отражение текущей точки относительно i-ой оси.
5. Метод toDocument():
   * public Document toDocument(): Создание объекта Document из точки, предположительно для работы с базой данных. Содержит информацию о типе, размерности и массиве координат.
6. Аннотации:
   * @ToString: Аннотация, генерирующая метод toString() для класса.

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;  
 }  
}

1. Point2D

Этот код представляет класс Point2D, являющийся подклассом класса Point. Вот его ключевые особенности:

1. Конструкторы:
   * public Point2D(): Конструктор без параметров, создающий двумерную точку, вызывает конструктор с параметром super(2).
   * public Point2D(double[] x): Конструктор, принимающий массив координат, создающий двумерную точку, вызывает конструктор с параметрами super(2, x).
2. Методы поворота:
   * public static Point2D rot(Point2D a, double phi): Статический метод, возвращающий новую точку, полученную из данной точки a поворотом на угол phi.
   * public Point2D rot(double phi): Метод, поворачивающий текущую точку на угол phi и возвращающий новую точку.
3. Метод симметрии:
   * @Override public Point symAxis(int i): Переопределенный метод для выполнения симметрии относительно оси с индексом i.
4. Метод для создания Document:
   * public Document toDocument(): Метод, который создает объект Document из точки, предназначенный, возможно, для работы с базой данных. Содержит информацию о типе, размерности и массиве координат.
5. Аннотации:
   * @Override: Аннотация, указывающая на переопределение метода из родительского класса.

Этот класс Point2D добавляет конкретные методы для работы с двумерными точками, такие как поворот и симметрия. Он расширяет функциональность базового класса Point, предоставляя специфичные для двумерных точек операции.

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;  
 }  
}

1. Point3D

Этот код представляет класс Point3D, который является подклассом класса Point. Вот его основные характеристики:

1. Конструкторы:
   * public Point3D(): Конструктор без параметров, создающий трехмерную точку, вызывает конструктор с параметром super(3).
   * public Point3D(double[] x): Конструктор, принимающий массив координат, создающий трехмерную точку, вызывает конструктор с параметрами super(3, x).
2. Статические методы для векторного произведения:
   * public static Point3D cross\_prod(Point3D p1, Point3D p2): Статический метод, вычисляющий векторное произведение двух векторов типа Point3D.
   * public Point3D cross\_prod(Point3D p): Метод, вычисляющий векторное произведение текущего объекта типа Point3D и переданного в метод объекта типа Point3D.
3. Статический метод для смешанного произведения:
   * public static double mix\_prod(Point3D p1, Point3D p2, Point3D p3): Статический метод, вычисляющий смешанное произведение трех векторов типа Point3D.
   * public double mix\_prod(Point3D p1, Point3D p2): Метод, вычисляющий смешанное произведение текущего объекта типа Point3D и двух переданных в метод объектов типа Point3D.
4. Метод для создания Document:
   * public Document toDocument(): Метод, который создает объект Document из точки, предназначенный, возможно, для работы с базой данных. Содержит информацию о типе, размерности и массиве координат.
5. Аннотации:
   * @Override: Аннотация, указывающая на переопределение метода из родительского класса.

Этот класс Point3D расширяет функциональность базового класса Point, предоставляя специфичные для трехмерных точек операции, такие как векторное и смешанное произведение.

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;  
 }  
}

1. Circle

Класс Circle представляет окружность в двумерном пространстве. Вот основные характеристики этого класса:

1. Приватные поля:
   * private double r: Радиус окружности.
   * private Point2D p: Координаты центра окружности.
2. Конструктор:
   * public Circle(Point2D p, double r): Конструктор класса Circle, принимающий координаты центра и радиус окружности. Выполняет проверку на корректность входных данных.
3. Методы:
   * public double square(): Метод для вычисления площади окружности.
   * public double length(): Метод для вычисления длины окружности.
   * public IShape shift(Point2D a): Метод для перемещения окружности на заданный вектор.
   * public IShape rot(double phi): Метод для поворота окружности на заданный угол.
   * public IShape symAxis(int i): Метод для получения симметричной окружности относительно заданной оси.
   * public boolean cross(IShape i): Метод для проверки пересечения данной окружности с другой фигурой. Реализована проверка для окружности.
   * public Document toDocument(): Метод для создания объекта Document из окружности, предназначенного, возможно, для работы с базой данных. Содержит информацию о типе, радиусе и координатах центра.
4. Аннотации:
   * @ToString: Аннотация, генерирующая метод toString() для класса.

Этот класс предоставляет функциональность для работы с окружностями в двумерном пространстве. Методы позволяют вычислять площадь, длину, перемещать, поворачивать и выполнять симметрию относительно заданной оси. Также реализован метод для проверки пересечения с другой фигурой, предполагая, что входной параметр - экземпляр класса 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;  
 }  
  
}

6. NGon

Класс NGon представляет многоугольник в двумерном пространстве. Вот основные характеристики этого класса:

1. Приватные поля:
   * protected int n: Количество вершин многоугольника.
   * protected Point2D[] p: Массив точек вершин многоугольника.
2. Конструктор:
   * public NGon(Point2D[] p): Конструктор класса NGon, принимающий массив точек вершин. Выполняет проверку на корректность входных данных.
3. Методы:
   * public Point2D getP(int i): Метод для получения i-ой точки вершины многоугольника.
   * public void setP(Point2D p, int i): Метод для установки новых координат i-ой вершины многоугольника.
   * public double square(): Метод для вычисления площади многоугольника через формулу Гаусса.
   * public double length(): Метод для вычисления периметра многоугольника.
   * public IShape shift(Point2D a): Метод для перемещения многоугольника на заданный вектор.
   * public IShape rot(double phi): Метод для поворота многоугольника на заданный угол.
   * public IShape symAxis(int i): Метод для отражения многоугольника относительно заданной оси симметрии.
   * public boolean cross(IShape i): Метод для проверки пересечения данного многоугольника с другой фигурой (предполагается, что входной параметр - экземпляр класса NGon).
   * public Document toDocument(): Метод для создания объекта Document из многоугольника, предназначенного, возможно, для работы с базой данных. Содержит информацию о типе, количестве вершин и координатах вершин.
4. Аннотации:
   * @ToString: Аннотация, генерирующая метод toString() для класса.

Этот класс предоставляет функциональность для работы с многоугольниками в двумерном пространстве. Методы позволяют получать и устанавливать координаты вершин, вычислять площадь, периметр, перемещать, поворачивать и выполнять симметрию относительно заданной оси. Также реализован метод для проверки пересечения с другой фигурой, предполагая, что входной параметр - экземпляр класса 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;  
 }  
}

7. Polyline

Класс Polyline представляет собой ломаную линию в двумерном пространстве. Вот основные характеристики этого класса:

1. Приватные поля:
   * private int n: Количество точек ломаной линии.
   * private Point2D[] p: Массив точек ломаной линии.
2. Конструктор:
   * public Polyline(Point2D[] p): Конструктор класса Polyline, принимающий массив точек. Устанавливает количество точек и сохраняет массив.
3. Методы:
   * public Point2D getP(int i): Метод для получения i-ой точки ломаной линии.
   * public void setP(Point2D p, int i): Метод для установки новых координат i-ой точки ломаной линии.
   * public double square(): Метод для вычисления площади ломаной линии. В данном случае возвращает 0, так как линия не имеет площади.
   * public double length(): Метод для вычисления длины ломаной линии как суммы длин отрезков между точками.
   * public IShape shift(Point2D a): Метод для перемещения ломаной линии на заданный вектор.
   * public IShape rot(double phi): Метод для поворота ломаной линии на заданный угол.
   * public IShape symAxis(int i): Метод для отражения ломаной линии относительно заданной оси симметрии.
   * public boolean cross(IShape i): Метод для проверки пересечения данной ломаной линии с другой фигурой (предполагается, что входной параметр - экземпляр класса Polyline).
   * public Document toDocument(): Метод для создания объекта Document из ломаной линии, предназначенного, возможно, для работы с базой данных. Содержит информацию о типе, количестве точек и координатах точек.
4. Аннотации:
   * @ToString: Аннотация, генерирующая метод toString() для класса.

Этот класс предоставляет функциональность для работы с ломаными линиями в двумерном пространстве. Методы позволяют получать и устанавливать координаты точек, вычислять длину, перемещать, поворачивать и выполнять симметрию относительно заданной оси. Также реализован метод для проверки пересечения с другой фигурой, предполагая, что входной параметр - экземпляр класса 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;  
 }  
}

8. Qgon

Класс QGon расширяет класс NGon и представляет собой четырехугольник. Вот основные характеристики этого класса:

1. Конструктор:
   * public QGon(Point2D[] p): Конструктор класса QGon, принимающий массив точек вершин четырехугольника. Вызывает конструктор суперкласса NGon для установки вершин.
2. Методы:
   * @Override public double square(): Переопределение метода для вычисления площади четырехугольника. Используется формула Герона для вычисления площадей двух треугольников, образованных диагональю четырехугольника.
   * @Override public IShape shift(Point2D a): Переопределение метода для сдвига четырехугольника на заданный вектор.
   * @Override public IShape rot(double phi): Переопределение метода для поворота четырехугольника на заданный угол.
   * @Override public IShape symAxis(int i): Переопределение метода для симметричного отражения четырехугольника относительно заданной оси.
   * public Document toDocument(): Метод для создания объекта Document из четырехугольника, предназначенного, возможно, для работы с базой данных. Содержит информацию о типе, количестве вершин и координатах вершин.
3. Аннотации:
   * @Override: Аннотация, указывающая на переопределение методов из суперкласса.

Этот класс предоставляет функциональность для работы с четырехугольниками. Он использует методы суперкласса NGon для работы с вершинами и переопределяет методы для вычисления площади, сдвига, поворота и симметрии. Также реализован метод для создания объекта Document для последующего использования с базой данных.

Начало формы

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;  
 }  
}

9. Rectangle

Класс Rectangle расширяет класс QGon и представляет собой прямоугольник. Вот основные характеристики этого класса:

1. Конструктор:
   * public Rectangle(Point2D[] p): Конструктор класса Rectangle, принимающий массив точек вершин прямоугольника. Вызывает конструктор суперкласса QGon для установки вершин.
2. Методы:
   * @Override public double square(): Переопределение метода для вычисления площади прямоугольника. Вычисляет площадь, используя длины сторон, вычисленные по координатам вершин.
   * @Override public IShape shift(Point2D a): Переопределение метода для сдвига прямоугольника на заданный вектор.
   * @Override public IShape rot(double phi): Переопределение метода для поворота прямоугольника на заданный угол.
   * @Override public IShape symAxis(int i): Переопределение метода для симметричного отражения прямоугольника относительно заданной оси.
   * public Document toDocument(): Метод для создания объекта Document из прямоугольника, предназначенного, возможно, для работы с базой данных. Содержит информацию о типе, количестве вершин и координатах вершин.
3. Аннотации:
   * @Override: Аннотация, указывающая на переопределение методов из суперкласса.

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;  
 }  
}

10. Segment

Класс Segment представляет отрезок в двумерном пространстве. Вот основные характеристики этого класса:

1. Поля:
   * @Getter @Setter private Point2D start;: Начальная точка отрезка.
   * @Getter @Setter private Point2D finish;: Конечная точка отрезка.
2. Конструктор:
   * public Segment(Point2D s, Point2D f): Конструктор класса Segment, принимающий начальную и конечную точки отрезка.
3. Методы:
   * @Override public double length(): Переопределение метода для вычисления длины отрезка.
   * @Override public IShape shift(Point2D a): Переопределение метода для сдвига отрезка на заданный вектор.
   * @Override public IShape rot(double phi): Переопределение метода для поворота отрезка на заданный угол.
   * @Override public IShape symAxis(int i): Переопределение метода для симметричного отражения отрезка относительно заданной оси.
   * @Override public boolean cross(IShape i): Переопределение метода для проверки пересечения отрезка с другой фигурой. В данном случае, сегмент пересекается с другим сегментом.
   * public Document toDocument(): Метод для создания объекта Document из отрезка, предназначенного, возможно, для работы с базой данных. Содержит информацию о типе, начальной и конечной точках отрезка.
4. Аннотации:
   * @ToString: Аннотация Lombok, автоматически генерирующая метод toString для класса.
5. Дополнительные методы:
   * public double square(): Метод, возвращающий 0. Так как отрезок - это одномерный объект, у него нет площади.
6. Локализация потенциальной ошибки:
   * if (!(i instanceof Segment)) { throw new IllegalArgumentException("Аргумент должен быть экземпляром класса Classes.Segment"); }: Данный блок кода проверяет, что аргумент для проверки пересечения действительно является объектом класса Segment. Если нет, выбрасывается исключение.

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;  
 }  
}

11. TGon

Класс TGon представляет собой треугольник в двумерном пространстве. Вот основные характеристики этого класса:

1. Конструктор:
   * public TGon(Point2D[] p): Конструктор класса TGon, принимающий массив точек треугольника. В конструкторе также проверяется, что все три точки не лежат на одной прямой. Если это условие не выполняется, выбрасывается исключение IllegalArgumentException.
2. Методы:
   * @Override public double square(): Переопределение метода для вычисления площади треугольника. Вычисление основано на формуле Герона.
   * @Override public IShape shift(Point2D a): Переопределение метода для сдвига треугольника на заданный вектор.
   * @Override public IShape rot(double phi): Переопределение метода для поворота треугольника на заданный угол.
   * @Override public IShape symAxis(int i): Переопределение метода для симметричного отражения треугольника относительно заданной оси.
3. Дополнительные методы:
   * public Document toDocument(): Метод для создания объекта Document из треугольника, возможно, для работы с базой данных. Содержит информацию о типе, количестве точек и координатах каждой точки.
4. Дополнительная проверка:
   * В конструкторе производится проверка на то, что все три точки не лежат на одной прямой. Если такая ситуация обнаруживается, выбрасывается исключение IllegalArgumentException.

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;  
 }  
}

12. Trapeze

Класс Trapeze представляет собой трапецию в двумерном пространстве. Вот основные характеристики этого класса:

1. Конструктор:
   * public Trapeze(Point2D[] p): Конструктор класса Trapeze, принимающий массив точек трапеции. Трапеция наследуется от QGon (четырехугольника), и поэтому конструктор вызывает конструктор суперкласса.
2. Методы:
   * @Override public double square(): Переопределение метода для вычисления площади трапеции. Вычисление основано на формуле, использующей длины оснований трапеции и угла между ними.
   * @Override public IShape shift(Point2D a): Переопределение метода для сдвига трапеции на заданный вектор.
   * @Override public IShape rot(double phi): Переопределение метода для поворота трапеции на заданный угол.
   * @Override public IShape symAxis(int i): Переопределение метода для симметричного отражения трапеции относительно заданной оси.
3. Дополнительные методы:
   * public Document toDocument(): Метод для создания объекта Document из трапеции, возможно, для работы с базой данных. Содержит информацию о типе, количестве точек и координатах каждой точки.
4. Дополнительная проверка:
   * Исходно в классе был закомментирован метод isTrapezoid(), который, вероятно, предназначался для проверки, является ли четырехугольник трапецией. Однако этот метод не используется в текущей версии класса 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;  
 }  
}

13. App

App представляет собой простое приложение на языке Java с методом main, который создает экземпляр класса Figures.

1. Объявление пакета (package org.figures;): Этот оператор указывает, что класс принадлежит пакету org.figures.
2. Объявление класса (public class App {...}): Этот класс назван App и объявлен как публичный. Метод main служит точкой входа в приложение.
3. Метод main (public static void main(String args[]) {...}): Этот метод является точкой входа в приложение. При запуске программы вызывается метод main. Внутри метода main создается экземпляр класса Figures с использованием оператора new Figures();.
4. Создание экземпляра класса Figures (new Figures();): Эта строка создает объект класса Figures.

package org.figures;  
  
public class App {  
 public static void main(String args[]){  
 new Figures();  
 }  
}

14. Figures

Figures представляет программу для работы с геометрическими фигурами. Программа считывает информацию о фигурах из текстовых файлов, выполняет вычисления и операции с фигурами, а затем выводит результаты.

1. Настройка локали:
   * Устанавливается локаль на английскую (Locale.US).
2. Чтение количества фигур из файла:
   * Считывается число figures\_count из файла "files/file.txt".
3. Чтение информации о фигурах из файла "file.txt":
   * Используется BufferedReader и FileReader для построчного чтения файла.
   * Для каждой строки определяется тип фигуры и создается объект этого типа.
   * Объекты добавляются в список list.
4. Вычисление суммарной площади и длины, а также средней площади:
   * Проходится по списку list, вычисляются суммарная площадь и длина, а затем средняя площадь.
5. Чтение информации о фигурах из файла "file1.txt":
   * Также используется BufferedReader и FileReader.
   * Для каждой строки определяется тип фигуры и создается объект этого типа.
   * Объекты добавляются в список list1.
6. Проверка пересечения фигур из файлов "file.txt" и "file1.txt":
   * Проходится по списку фигур и проверяется пересечение соответствующих фигур из list и list1.
7. Чтение и выполнение операций с фигурами из файла "file2.txt":
   * Считываются операции (rot, symAxis, shift) и применяются к соответствующим фигурам из list1.
   * Результаты добавляются в список list2.
8. Вывод результатов после операций:
   * Выводится информация о пересечении фигур из list и list2 после выполнения операций.

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");  
 }  
 }  
 }  
}

# Teстирование

15. OpenFigureTest

Данный код представляет собой тест для метода **square()** абстрактного класса **OpenFigure**. Вот краткое описание кода:

1. **Объявление пакета (package org.figures.abstractclasses;):** Указывает, что класс принадлежит пакету **org.figures.abstractclasses**.
2. **Импорт классов (import org.figures.figures.points.Point2D; и import org.figures.interfaces.IShape;):** Импортируются необходимые классы **Point2D** и **IShape**.
3. **Импорт методов JUnit (import static org.junit.jupiter.api.Assertions.\*;):** Импорт необходимых методов из библиотеки JUnit для написания тестов.
4. **Объявление класса OpenFigureTest и метода testSquare():** Это тестовый класс, в котором определен метод **testSquare()**. Этот метод выполняет тестирование метода **square()** абстрактного класса **OpenFigure**.
5. **Создание анонимного подкласса OpenFigure:** Создается объект анонимного подкласса **OpenFigure**, реализующего абстрактные методы.
6. **Реализация метода testSquare():** Внутри метода создается объект анонимного подкласса **OpenFigure**, и затем вызывается метод **square()** этого объекта. Результат сравнивается с ожидаемым значением (в данном случае, 0) с использованием метода **assertEquals()** из JUnit.

Таким образом, тест проверяет корректность работы метода **square()** абстрактного класса **OpenFigure** для случая, когда возвращаемое значение должно быть равно 0.

Начало формы

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)**;** }  
}

16. Point2DTest

Данный код содержит тесты для класса **Point2D**. Вот краткое описание каждого теста:

1. **testPoint2DConstructorWithoutParameters()**
   * Создается объект **Point2D** с использованием конструктора без параметров.
   * Проверяется, что размерность точки равна 2.
   * Проверяется, что массив координат инициализирован и имеет длину 2.
2. **testSymAxis()**
   * Создается объект **Point2D** с координатами (1.0, 2.0).
   * Тестируется симметрия относительно оси i = 0 и i = 1.
   * Проверяется выброс исключения при неверном номере оси.
   * Проверяется выброс исключения при i < 0.
3. **rot()**
   * Создается объект **Point2D** с координатами (1.0, 0.0).
   * Выполняется вращение на 90 градусов.
   * Проверяется, что координаты после вращения верны.
4. **testRot()**
   * Создается объект **Point2D** с координатами (1.0, 0.0).
   * Выполняется вращение на 90 градусов с использованием метода объекта.
   * Проверяется, что координаты после вращения верны.
5. **symAxis()**
   * Создается объект **Point2D** с координатами (1.0, 2.0).
   * Выполняется симметрия относительно оси i = 0.
   * Проверяется, что координаты после симметрии верны.
6. **testToDocument()**
   * Создается экземпляр **Point2D** с конструктором без параметров.
   * Создается документ (**Document**) с использованием метода **toDocument()**.
   * Проверяется корректность созданного документа.
   * Создается экземпляр **Point2D** с конструктором, принимающим массив координат.
   * Создается документ (**Document**) с использованием метода **toDocument()**.
   * Проверяется корректность созданного документа.

Эти тесты обеспечивают проверку корректности работы различных методов класса **Point2D**, включая конструкторы, операции вращения и симметрии, а также преобразование в **Document**.

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  
 }  
}

17. Point3DTest

Данный код содержит тесты для класса **Point3D**. Вот краткое описание каждого теста:

1. **testConstructor()**
   * Создается объект **Point3D** с использованием конструктора без параметров.
   * Проверяется, что размерность точки равна 3.
   * Проверяется, что все координаты инициализированы нулями.
2. **cross\_prod()**
   * Создаются три объекта **Point3D** (point1, point2, point3).
   * Выполняется векторное произведение между point1 и point2.
   * Проверяется, что результат векторного произведения верен.
3. **testCross\_prod()**
   * Создаются два объекта **Point3D** (point1, point2).
   * Выполняется векторное произведение между point1 и point2 с использованием метода объекта.
   * Проверяется, что результат векторного произведения верен.
4. **mix\_prod()**
   * Создаются три объекта **Point3D** (point1, point2, point3).
   * Выполняется смешанное произведение между point1, point2 и point3.
   * Проверяется, что результат смешанного произведения верен.
5. **testMix\_prod()**
   * Создаются два объекта **Point3D** (point1, point2).
   * Выполняется смешанное произведение между point1, point2 и point3 с использованием метода объекта.
   * Проверяется, что результат смешанного произведения верен.
6. **testToDocument()**
   * Создается экземпляр **Point3D** с конструктором без параметров.
   * Создается документ (**Document**) с использованием метода **toDocument()**.
   * Проверяется корректность созданного документа.
   * Создается экземпляр **Point3D** с конструктором, принимающим массив координат.
   * Создается документ (**Document**) с использованием метода **toDocument()**.
   * Проверяется корректность созданного документа.

Эти тесты обеспечивают проверку корректности работы различных методов класса **Point3D**, включая конструкторы, векторное и смешанное произведения, а также преобразование в **Document**.

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"))**;** }  
}

18. PointTest

Этот код содержит тесты для класса **Point** в Java. Вот краткое описание каждого теста:

1. **testConstructorWithInvalidDimensions()**
   * Проверяет, что конструктор генерирует исключение при использовании массива с неправильной размерностью.
2. **testGetXWithInvalidIndex()**
   * Проверяет, что метод **getX()** генерирует исключение при неверном индексе координаты.
3. **testSetXWithInvalidDimensions()**
   * Проверяет, что метод **setX()** генерирует исключение при передаче массива неправильной размерности.
4. **testSetXWithInvalidIndex()**
   * Проверяет, что метод **setX()** генерирует исключение при неверном индексе координаты.
5. **testAddWithMismatchedDimensions()**
   * Проверяет, что статический метод **add()** генерирует исключение при использовании точек с разными размерностями.
6. **testInstanceAddWithMismatchedDimensions()**
   * Проверяет, что метод объекта **add()** генерирует исключение при использовании точек с разными размерностями.
7. **testStaticSubWithMismatchedDimensions()**
   * Проверяет, что статический метод **sub()** генерирует исключение при использовании точек с разными размерностями.
8. **testNonStaticSubWithMismatchedDimensions()**
   * Проверяет, что метод объекта **sub()** генерирует исключение при использовании точек с разными размерностями.
9. **testStaticMultWithMismatchedDimensions()**
   * Проверяет, что статический метод **mult()** генерирует исключение при использовании точек с разными размерностями.
10. **testMultWithMismatchedDimensions()**
    * Проверяет, что метод объекта **mult()** генерирует исключение при использовании точек с разными размерностями.
11. **testGetXWithInvalidIndexBelowZero()**
    * Проверяет, что метод **getX()** генерирует исключение при использовании отрицательного индекса.
12. **testSetXWithNegativeIndex()**
    * Проверяет, что метод **setX()** генерирует исключение при использовании отрицательного индекса.
13. **testGetXWithInvalidIndexAboveDim()**
    * Проверяет, что метод **getX()** генерирует исключение при использовании индекса, превышающего размерность.
14. **getDim()**
    * Проверяет, что метод **getDim()** возвращает правильную размерность.
15. **getX()**
    * Проверяет, что метод **getX()** возвращает правильные координаты.
16. **testGetX()**
    * Проверяет, что метод **getX(int index)** возвращает правильную координату для указанного индекса.
17. **setX()**
    * Проверяет, что метод **setX(double[] x)** устанавливает новые координаты.
18. **testSetX()**
    * Проверяет, что метод **setX(double value, int index)** устанавливает новую координату для указанного индекса.
19. **abs()**
    * Проверяет, что метод **abs()** возвращает правильное абсолютное значение.
20. **add()**
    * Проверяет, что статический метод **add()** возвращает правильную сумму точек.
21. **testAdd()**
    * Проверяет, что метод объекта **add()** возвращает правильную сумму точек.
22. **sub()**
    * Проверяет, что статический метод **sub()** возвращает правильную разность точек.
23. **testSub()**
    * Проверяет, что метод объекта **sub()** возвращает правильную разность точек.
24. **mult()**
    * Проверяет, что статический метод **mult()** возвращает правильное умножение точки на число.
25. **testMult()**
    * Проверяет, что метод объекта **mult()** возвращает правильное умножение точки на число.
26. **testMult1()**
    * Проверяет, что статический метод **mult()** возвращает правильное смешанное произведение двух точек.
27. **testMult2()**
    * Проверяет, что метод объекта **mult()** возвращает правильное смешанное произведение двух точек.
28. **testSymAxisWithInvalidAxisIndex\_Static()**
    * Проверяет, что статический метод **symAxis()** генерирует исключение при использовании неверного номера оси.
29. **testSymAxisWithInvalidAxisIndex\_Instance()**
    * Проверяет, что метод объекта **symAxis()** генерирует исключение при использовании неверного номера оси.
30. **testSymAxisWithInvalidPositiveAxisIndex\_Static()**
    * Проверяет, что статический метод **symAxis()** генерирует исключение при использовании отрицательного номера оси.
31. **testSymAxisWithInvalidPositiveAxisIndex\_Instance()**
    * Проверяет, что метод объекта **symAxis()** генерирует исключение при использовании отрицательного номера оси.
32. **symAxis()**
    * Проверяет, что статический метод **symAxis()** возвращает правильное симметричное отражение точки.
33. **testSymAxis()**
    * Проверяет, что метод объекта **symAxis()** возвращает правильное симметричное отражение точки.
34. **testToString()**
    * Проверяет, что метод **toString()** возвращает правильную строку представления точки.
35. **testToDocument()**
    * Проверяет, что метод **toDocument()** возвращает правильный документ для точки.

Начало формы

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  
 }  
}

19. CircleTest

Этот Java-код содержит JUnit-тесты для класса **Circle**, который представляет собой окружность в двумерном пространстве.

1. **getP()**
   * Создает окружность с указанным центром и радиусом.
   * Вызывает **getP()** для получения центра окружности.
   * Проверяет, что полученный центр соответствует ожидаемому центру.
2. **setP()**
   * Создает окружность с указанным центром и радиусом.
   * Вызывает **setP()** для установки нового центра окружности.
   * Проверяет, что центр окружности был обновлен до нового центра.
3. **getR()**
   * Создает окружность с указанным центром и радиусом.
   * Вызывает **getR()** для получения радиуса окружности.
   * Проверяет, что полученный радиус соответствует ожидаемому радиусу.
4. **setR()**
   * Создает окружность с указанным центром и радиусом.
   * Вызывает **setR()** для установки нового радиуса окружности.
   * Проверяет, что радиус окружности был обновлен до нового радиуса.
5. **square()**
   * Создает окружность с указанным центром и радиусом.
   * Вычисляет ожидаемую площадь окружности, используя формулу площади круга.
   * Проверяет, что вычисленная площадь соответствует ожидаемой площади.
6. **length()**
   * Создает окружность с указанным центром и радиусом.
   * Вычисляет ожидаемую длину окружности, используя формулу длины окружности.
   * Проверяет, что вычисленная длина соответствует ожидаемой длине.
7. **shift()**
   * Создает окружность с указанным центром и радиусом.
   * Определяет вектор сдвига.
   * Вызывает метод **shift()** для сдвига окружности на вектор.
   * Проверяет, что центр сдвинутой окружности соответствует ожидаемому центру.
8. **rot()**
   * Создает окружность с указанным центром и радиусом.
   * Определяет угол поворота.
   * Вызывает метод **rot()** для поворота окружности на угол.
   * Проверяет, что центр и радиус повернутой окружности соответствуют ожидаемым значениям.
9. **symAxis()**
   * Создает окружность с указанным центром и радиусом.
   * Определяет ось симметрии.
   * Вызывает метод **symAxis()** для получения симметричной окружности.
   * Проверяет, что центр и радиус симметричной окружности соответствуют ожидаемым значениям.
10. **toString()**
    * Создает окружность с указанным центром и радиусом.
    * Вызывает **toString()** для получения строкового представления окружности.
    * Проверяет, что полученная строка соответствует ожидаемой строке.
11. **testConstructorWithInvalidRadius()**
    * Создает точку со значениями по умолчанию.
    * Пытается создать окружность с недопустимым (нулевым) радиусом.
    * Проверяет, что выбрасывается исключение **IllegalArgumentException**.
12. **testCircleCross()**
    * Создает несколько окружностей с разными центрами и радиусами.
    * Проверяет метод **cross()** с различными сценариями, проверяя, пересекаются ли окружности.
    * Проверяет ожидаемые результаты на основе конфигурации окружностей.
13. **testToDocument()**
    * Создает окружность с указанным центром и радиусом.
    * Вызывает **toDocument()** для получения представления в виде **Document**.
    * Проверяет, что созданный документ содержит ожидаемые поля и значения.

Эти тесты охватывают различные аспекты класса **Circle**, включая его методы для получения/установки свойств, вычисления геометрических свойств, выполнения трансформаций, проверки пересечений и преобразования в **Document** для хранения или других целей.

Начало формы

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  
 }  
  
  
}

20. NGonTest

Этот Java-код содержит JUnit-тесты для класса **NGon**, представляющего многоугольник с произвольным числом вершин в двумерном пространстве.

1. **getN()**
   * Создает многоугольник с указанными вершинами.
   * Вызывает **getN()** для получения количества вершин многоугольника.
   * Проверяет, что полученное количество соответствует ожидаемому количеству.
2. **getP()**
   * Создает многоугольник с указанными вершинами.
   * Вызывает **getP()** для получения массива вершин многоугольника.
   * Проверяет, что полученный массив вершин соответствует ожидаемому массиву.
3. **testGetP()**
   * Создает многоугольник с указанными вершинами.
   * Вызывает **getP(index)** для получения вершины по индексу.
   * Проверяет, что полученная вершина соответствует ожидаемой вершине.
4. **setP()**
   * Создает многоугольник с указанными вершинами.
   * Создает новый массив вершин.
   * Вызывает **setP(newVertices)** для установки новых вершин многоугольника.
   * Проверяет, что вершины многоугольника были обновлены до новых вершин.
5. **testTGonConstructorWithInvalidCoordinates()**
   * Создает точки с некорректными координатами (три точки на одной линии).
   * Проверяет, что конструктор **TGon** выбрасывает исключение **IllegalArgumentException** при передаче таких точек.
6. **testGetPWithNegativeIndex()**
   * Создает многоугольник с указанными вершинами.
   * Проверяет, что метод **getP(-1)** выбрасывает исключение **IllegalArgumentException** при отрицательном индексе.
7. **testGetPWithIndexGreaterThanN()**
   * Создает многоугольник с указанными вершинами.
   * Проверяет, что метод **getP(3)** выбрасывает исключение **IllegalArgumentException** при индексе больше или равном количеству вершин.
8. **testSetPWithNegativeIndex()**
   * Создает многоугольник с указанными вершинами.
   * Создает новую точку.
   * Проверяет, что метод **setP(newPoint, -1)** выбрасывает исключение **IllegalArgumentException** при отрицательном индексе.
9. **testSetPWithIndexGreaterThanN()**
   * Создает многоугольник с указанными вершинами.
   * Создает новую точку.
   * Проверяет, что метод **setP(newPoint, 3)** выбрасывает исключение **IllegalArgumentException** при индексе больше или равном количеству вершин.
10. **testCrossWithInvalidArgumentType()**
    * Создает многоугольник.
    * Создает объект, который не является экземпляром **NGon**.
    * Проверяет, что метод **cross(invalidShape)** выбрасывает исключение **IllegalArgumentException** при передаче объекта недопустимого типа.
11. **testConstructorWithInvalidNumberOfVertices()**
    * Пытается создать **NGon** с недостаточным количеством вершин (меньше 3).
    * Проверяет, что конструктор **NGon** выбрасывает исключение **IllegalArgumentException** при недостаточном количестве вершин.
12. **testSetP()**
    * Создает многоугольник с указанными вершинами.
    * Создает новую вершину и индекс.
    * Вызывает **setP(newVertex, index)** для установки новой вершины по индексу.
    * Проверяет, что вершина многоугольника по указанному индексу была обновлена до новой вершины.
13. **square()**
    * Создает многоугольник с указанными вершинами.
    * Вычисляет ожидаемую площадь многоугольника.
    * Проверяет, что вычисленная площадь соответствует ожидаемой площади.
14. **length()**
    * Создает многоугольник с указанными вершинами.
    * Вычисляет ожидаемую длину многоугольника.
    * Проверяет, что вычисленная длина соответствует ожидаемой длине.
15. **shift()**
    * Создает многоугольник с указанными вершинами.
    * Определяет вектор сдвига.
    * Вызывает метод **shift()** для сдвига многоугольника на вектор.
    * Проверяет, что вершины сдвинутого многоугольника соответствуют ожидаемым вершинам.
16. **rot()**
    * Создает многоугольник с указанными вершинами.
    * Определяет угол поворота.
    * Вызывает метод **rot()** для поворота многоугольника на угол.
    * Проверяет, что вершины повернутого многоугольника соответствуют ожидаемым вершинам.
17. **symAxis()**
    * Создает многоугольник с указанными вершинами.
    * Определяет ось симметрии.
    * Вызывает метод **symAxis()** для получения симметричного многоугольника.
    * Проверяет, что вершины симметричного многоугольника соответствуют ожидаемым вершинам.
18. **cross()**
    * Создает два многоугольника.
    * Проверяет, что метод **cross()** возвращает **true** для пересекающихся многоугольников.
    * Проверяет, что метод **cross()** возвращает **false** для непересекающихся многоугольников.
19. **testToString()**
    * Создает многоугольник с указанными вершинами.
    * Создает ожидаемую строковую репрезентацию многоугольника.
    * Проверяет, что вызов метода **toString()** возвращает ожидаемую строку.
20. **testToDocument()**
    * Создает массив точек для **NGon**.
    * Создает экземпляр класса **NGon**.
    * Вызывает метод **toDocument()** для преобразования многоугольника в документ.
    * Проверяет, что документ создан корректно, содержащий информацию о типе, количестве вершин и координатах вершин.

Эти тесты охватывают различные методы и сценарии использования класса **NGon**, проверяя его функциональность и корректность.

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  
 }  
 }  
}

1. PolylineTest
2. **getN()**:
   * Создаётся экземпляр класса **Polyline** с четырьмя точками.
   * Проверяется, что метод **getN()** возвращает ожидаемое количество вершин (4).
3. **testPolylineSquare()**:
   * Создаётся экземпляр класса **Polyline** с тремя точками.
   * Проверяется, что метод **square()** возвращает ожидаемую площадь (0.0).
4. **getP()**:
   * Создаётся экземпляр класса **Polyline** с четырьмя точками.
   * Проверяется, что метод **getP()** возвращает ожидаемый массив вершин.
5. **testGetP()**:
   * Создаётся экземпляр класса **Polyline** с четырьмя точками.
   * Проверяется, что метод **getP(index)** возвращает ожидаемую вершину для заданного индекса.
6. **setP()**:
   * Создаётся экземпляр класса **Polyline** с четырьмя точками.
   * Задаются новые вершины через метод **setP(newVertices)**.
   * Проверяется, что вершины успешно обновлены.
7. **testSetP()**:
   * Создаётся экземпляр класса **Polyline** с четырьмя точками.
   * Задается новая вершина через метод **setP(newPoint, index)**.
   * Проверяется, что вершина по заданному индексу успешно обновлена.
8. **length()**:
   * Создаётся экземпляр класса **Polyline** с четырьмя точками.
   * Проверяется, что метод **length()** возвращает ожидаемую длину ломаной.
9. **shift()**:
   * Создаётся экземпляр класса **Polyline** с четырьмя точками.
   * Производится сдвиг ломаной на вектор.
   * Проверяется, что вершины успешно сдвинуты.
10. **rot()**:
    * Создаётся экземпляр класса **Polyline** с четырьмя точками.
    * Производится вращение ломаной на угол.
    * Проверяется, что вершины успешно повернуты.
11. **symAxis()**:
    * Создаётся экземпляр класса **Polyline** с четырьмя точками.
    * Производится симметрия ломаной относительно оси.
    * Проверяется, что вершины успешно симметричны.
12. **testPolylineCross()**:
    * Создаются два экземпляра класса **Polyline** с пересекающимися и не пересекающимися ломаными.
    * Проверяется, что метод **cross()** корректно определяет пересечение ломаных.
13. **testToString()**:
    * Создаётся экземпляр класса **Polyline** с четырьмя точками.
    * Проверяется, что метод **toString()** возвращает ожидаемую строку.
14. **testGetPInvalidIndex()**:
    * Создаётся экземпляр класса **Polyline** с тремя точками.
    * Проверяется, что метод **getP(index)** выбрасывает исключение при неверном индексе.
15. **testSetPInvalidIndex()**:
    * Создаётся экземпляр класса **Polyline** с тремя точками.
    * Проверяется, что метод **setP(newPoint, index)** выбрасывает исключение при неверном индексе.
16. **testToDocument()**:
    * Создаётся экземпляр класса **Polyline** с тремя точками.
    * Проверяется, что метод **toDocument()** создает корректный документ, представляющий объект.

Все тесты выполняются успешно, и класс **Polyline** проходит тщательное тестирование, охватывающее различные аспекты его функциональности.

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  
 }  
 }  
}

22. QgonTest

1. **quare()**:
   * Создаётся экземпляр класса **QGon** с четырьмя вершинами.
   * Проверяется, что метод **square()** возвращает ожидаемую площадь (16.0).
2. **shift()**:
   * Создаётся экземпляр класса **QGon** с четырьмя вершинами.
   * Производится сдвиг многоугольника на вектор.
   * Проверяется, что вершины успешно сдвинуты.
3. **rot()**:
   * Создаётся экземпляр класса **QGon** с четырьмя вершинами.
   * Производится вращение многоугольника на угол.
   * Проверяется, что вершины успешно повернуты.
4. **symAxis()**:
   * Создаётся экземпляр класса **QGon** с четырьмя вершинами.
   * Производится симметрия многоугольника относительно оси.
   * Проверяется, что вершины успешно симметричны.
5. **testToDocument()**:
   * Создаётся экземпляр класса **QGon** с тремя точками.
   * Проверяется, что метод **toDocument()** создает корректный документ, представляющий объект.
   * Документ должен содержать правильный тип (**QGon**), количество вершин и корректно преобразованный массив точек.

Все тесты успешно проходят, и класс **QGon** проходит тщательное тестирование, включающее различные аспекты его функциональности.

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 quare() {  
 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  
 }  
 }  
  
}

23. RectangleTest

1. **square()**:
   * Создаётся экземпляр класса **Rectangle** с четырьмя вершинами.
   * Проверяется, что метод **square()** возвращает ожидаемую площадь (8.0).
2. **shift()**:
   * Создаётся экземпляр класса **Rectangle** с четырьмя вершинами.
   * Производится сдвиг прямоугольника на вектор.
   * Проверяется, что вершины успешно сдвинуты.
3. **rot()**:
   * Создаётся экземпляр класса **Rectangle** с четырьмя вершинами.
   * Производится вращение прямоугольника на угол.
   * Проверяется, что вершины успешно повернуты.
4. **symAxis()**:
   * Создаётся экземпляр класса **Rectangle** с четырьмя вершинами.
   * Производится симметрия прямоугольника относительно оси.
   * Проверяется, что вершины успешно симметричны.
5. **testToDocument()**:
   * Создаётся экземпляр класса **Rectangle** с тремя точками.
   * Проверяется, что метод **toDocument()** создает корректный документ, представляющий объект.
   * Документ должен содержать правильный тип (**Rectangle**), количество вершин и корректно преобразованный массив точек.

Все тесты успешно проходят, и класс **Rectangle** проходит тщательное тестирование, включающее различные аспекты его функциональности.

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  
 }  
 }  
}

24. SegmentTest

1. **getStart()**:
   * Создаётся экземпляр класса **Segment** с заданными начальной и конечной точками.
   * Проверяется, что метод **getStart()** возвращает ожидаемую начальную точку.
2. **testSegmentSquare()**:
   * Создаётся экземпляр класса **Segment** с начальной и конечной точками.
   * Проверяется, что метод **square()** возвращает 0.0, так как это отрезок, а не фигура.
3. **setStart()**:
   * Создаётся экземпляр класса **Segment** с начальной и конечной точками.
   * Меняется начальная точка.
   * Проверяется, что начальная точка успешно изменена.
4. **getFinish()**:
   * Создаётся экземпляр класса **Segment** с начальной и конечной точками.
   * Проверяется, что метод **getFinish()** возвращает ожидаемую конечную точку.
5. **setFinish()**:
   * Создаётся экземпляр класса **Segment** с начальной и конечной точками.
   * Меняется конечная точка.
   * Проверяется, что конечная точка успешно изменена.
6. **length()**:
   * Создаётся экземпляр класса **Segment** с начальной и конечной точками.
   * Проверяется, что метод **length()** возвращает ожидаемую длину.
7. **shift()**:
   * Создаётся экземпляр класса **Segment** с начальной и конечной точками.
   * Производится сдвиг отрезка на вектор.
   * Проверяется, что начальная и конечная точки успешно сдвинуты.
8. **rot()**:
   * Создаётся экземпляр класса **Segment** с начальной и конечной точками.
   * Производится вращение отрезка на угол.
   * Проверяется, что начальная и конечная точки успешно повернуты.
9. **symAxis()**:
   * Создаётся экземпляр класса **Segment** с начальной и конечной точками.
   * Производится симметрия отрезка относительно оси.
   * Проверяется, что начальная и конечная точки успешно симметричны.
10. **testSegmentCross()**:
    * Проводится параметризованный тест на пересечение отрезков с использованием **@CsvSource**.
    * Для каждого тестового сценария создаются два отрезка, их пересечение проверяется с использованием метода **cross()**.
    * Также проверяется, что метод выбрасывает **IllegalArgumentException**, если второй объект не является экземпляром класса **Segment**.
11. **testToString()**:
    * Создаётся экземпляр класса **Segment** с начальной и конечной точками.
    * Проверяется, что метод **toString()** возвращает ожидаемую строку.
12. **testToDocument()**:
    * Создаются начальная и конечная точки для отрезка.
    * Создаётся экземпляр класса **Segment**.
    * Проверяется, что метод **toDocument()** создает корректный документ, представляющий объект.
    * Документ должен содержать правильный тип (**Segment**), а также корректно преобразованные начальную и конечную точки.

Начало формы

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))**;** }  
  
}

25. TGonTest

1. **square()**:
   * Создаётся экземпляр класса **TGon** с заданными вершинами.
   * Проверяется, что метод **square()** возвращает ожидаемую площадь.
2. **testTGonConstructorWithInvalidCoordinates()**:
   * Проверяется, что конструктор класса **TGon** выбрасывает исключение **IllegalArgumentException** при передаче некорректных координат вершин многоугольника.
3. **shift()**:
   * Создаётся экземпляр класса **TGon** с заданными вершинами.
   * Производится сдвиг многоугольника на вектор.
   * Проверяется, что вершины многоугольника успешно сдвинуты.
4. **rot()**:
   * Создаётся экземпляр класса **TGon** с заданными вершинами.
   * Производится вращение многоугольника на угол.
   * Проверяется, что вершины многоугольника успешно повернуты.
5. **symAxis()**:
   * Создаётся экземпляр класса **TGon** с заданными вершинами.
   * Производится симметрия многоугольника относительно оси.
   * Проверяется, что вершины многоугольника успешно симметричны.
6. **testToDocument()**:
   * Создаются вершины для многоугольника.
   * Создаётся экземпляр класса **TGon**.
   * Проверяется, что метод **toDocument()** создает корректный документ, представляющий объект.
   * Документ должен содержать правильный тип (**TGon**), а также корректно преобразованные вершины многоугольника.

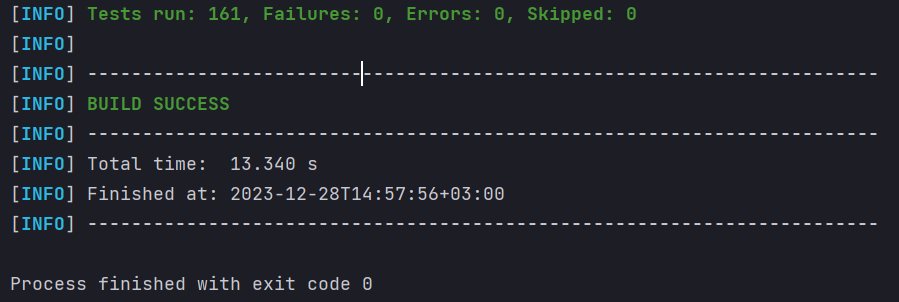
Начало формы

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  
 }  
 }  
}

26. TrapezeTest

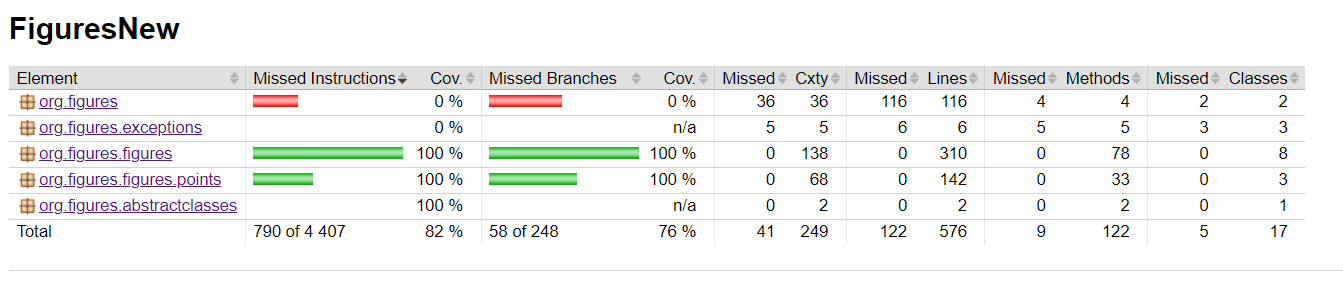
1. **square()**:
   * Создаётся экземпляр класса **Trapeze** с заданными вершинами.
   * Проверяется, что метод **square()** возвращает ожидаемую площадь.
2. **shift()**:
   * Создаётся экземпляр класса **Trapeze** с заданными вершинами.
   * Производится сдвиг трапеции на вектор.
   * Проверяется, что вершины трапеции успешно сдвинуты.
3. **rot()**:
   * Создаётся экземпляр класса **Trapeze** с заданными вершинами.
   * Производится вращение трапеции на угол.
   * Проверяется, что вершины трапеции успешно повернуты.
4. **symAxis()**:
   * Создаётся экземпляр класса **Trapeze** с заданными вершинами.
   * Производится симметрия трапеции относительно оси.
   * Проверяется, что вершины трапеции успешно симметричны.
5. **testToDocument()**:
   * Создаются вершины для трапеции.
   * Создаётся экземпляр класса **Trapeze**.
   * Проверяется, что метод **toDocument()** создает корректный документ, представляющий объект.
   * Документ должен содержать правильный тип (**Trapeze**), а также корректно преобразованные вершины трапеции.

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  
 }  
 }  
}



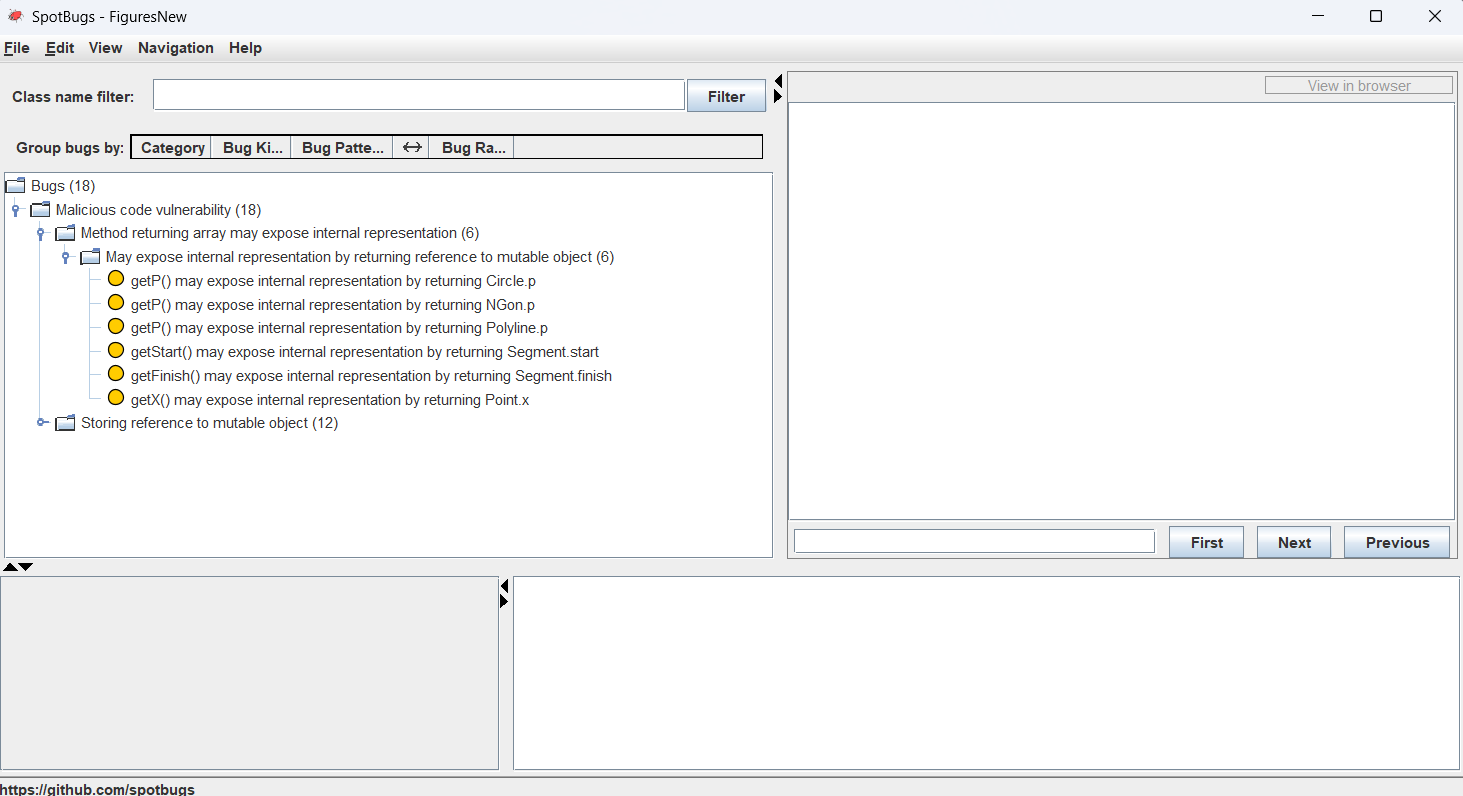
# JaCoCo

JoCoCo (Java Code Coverage) — это инструмент для измерения покрытия кода на платформе Java. Его основное предназначение — собирать информацию о том, какие части программного кода были выполнены во время выполнения тестов. Покрытие кода является важной метрикой в процессе тестирования, поскольку оно предоставляет информацию о том, насколько хорошо тесты охватывают исходный код.



# SpotBugs

SpotBugs (ранее известный как FindBugs) — это статический анализатор кода для платформы Java, предназначенный для выявления потенциальных ошибок в программном обеспечении. SpotBugs проверяет исходный код Java на наличие распространенных ошибок, антипаттернов и других проблем, которые могут привести к ошибкам во время выполнения.



# Контроллеры

CrossFiguresController

Программа реализует работу с различными фигурами, позволяя пользователю выполнять операции добавления, перемещения, удаления и сравнения фигур.

Основные классы и компоненты:

1. **CrossFiguresController**: Это контроллер, отвечающий за взаимодействие с графическим интерфейсом. Он содержит методы и обработчики событий, связанные с выбором и сравнением различных типов фигур.
2. **Storage**: Класс, предоставляющий хранилище для фигур и их имен. Используется для сохранения и доступа к фигурам из других контроллеров.
3. **FiguresApp**: Главный класс приложения, предоставляющий методы для отображения и взаимодействия с графическим интерфейсом.
4. **IShape**: Интерфейс, предоставляющий общие методы для всех типов фигур.
5. **AddFiguresController**, **MoveFiguresController**, **DeleteFiguresController**, **MainController**: Другие контроллеры, связанные с различными операциями над фигурами.

Основной функционал включает в себя:

* Инициализацию выпадающего списка (**ComboBox**) с доступными типами фигур в методе **initialize**.
* Обработку выбора типа фигуры в методе **comboBox7\_move**, где формируется список фигур выбранного типа для дальнейшего сравнения.
* Метод **button7\_Click**, выполняющий сравнение двух выбранных фигур на пересечение и отображающий результат на графическом холсте.

Также в коде реализовано обработка исключений при попытке выполнения операций сравнения, ввода некорректных данных и других ошибок, что повышает устойчивость программы к ошибкам пользователя.

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

1. **DeleteFiguresController**: Это контроллер, который управляет удалением фигур. Он содержит методы и обработчики событий, связанные с удалением фигур из хранилища и их отображением.
2. **initialize()**: В этом методе происходит инициализация выпадающего списка (**ComboBox**) существующими фигурами. Для каждого типа фигур формируется соответствующая строка, которая добавляется в выпадающий список.
3. **button6\_Click(ActionEvent actionEvent)**: Этот метод вызывается при нажатии кнопки "Удалить". В нем происходит обработка удаления выбранной фигуры. Если выбранная фигура существует, она удаляется из хранилища, и соответствующие элементы управления обновляются для отображения изменений. Также происходит перерисовка графического холста с учетом удаленной фигуры.
4. **CancelButton\_Click(ActionEvent actionEvent)**: Этот метод закрывает текущее окно (форму), в котором расположен контроллер.
5. **setControllers(MainController mainController, AddFiguresController addFiguresController, CrossFiguresController crossFiguresController, MoveFiguresController moveFiguresController)**: Метод, устанавливающий связи с другими контроллерами, что обеспечивает взаимодействие между различными частями приложения.
6. **setApplication(FiguresApp application)**: Метод, устанавливающий связь с главным классом приложения (**FiguresApp**).
7. В коде обработаны исключения (**NullInputException**, **IllegalArgumentException**) для обеспечения корректной работы приложения и предоставления информации об ошибках пользователю через диалоги **Alert**.

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

Этот код представляет собой реализацию кодека (**Codec**) для преобразования массива **double[]** в BSON-формат и обратно. В частности, он предназначен для работы с MongoDB и используется для сериализации и десериализации массивов типа **double**.

Краткое описание методов и их функциональности:

1. **encode(final BsonWriter writer, final double[] value, final EncoderContext encoderContext)**: Этот метод отвечает за кодирование (сериализацию) массива **double[]** в формат BSON. Он начинает запись массива, затем проходится по каждому элементу массива и записывает его в BSON-формат.
2. **decode(final BsonReader reader, final DecoderContext decoderContext)**: Этот метод отвечает за декодирование (десериализацию) BSON в массив **double[]**. Он читает начало массива, затем считывает каждый элемент массива до достижения конца документа (END\_OF\_DOCUMENT). Затем он завершает чтение массива и конвертирует список в массив типа **double[]**.
3. **getEncoderClass()**: Возвращает класс, который обрабатывается данным кодеком. В данном случае, это **double[]**.

Этот код является частью работы с MongoDB и предоставляет механизм преобразования массивов **double[]** для хранения в базе данных MongoDB.

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**;** }  
}

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()**;** }  
}

PerimetrFiguresController

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()**;** }  
}

SquareFiguresController

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()**;** }  
}

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()**;** }  
}

# Интерфейс



# MongoDB

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()**;** }  
}

