**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ**

**УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

ЛАБОРАТОРНАЯ РАБОТА №5

по дисциплине: **«Визуальные средства разработки программных**

**приложений»**

на тему: **Разработка приложений с графическим интерфейсом в *Java***

Выполнил: студент гр. ИТП-31

Дашкевич Д.А.

Принял: ассистент

Белявский Е. В.

Гомель 2019

**Цель**: изучить работу с графическим интерфейсом в *Java*.

**Ход работы**

**Вариант 2**

# **Задание:** определить классы *Triangle* и *PolyAngle*. Определить, какой из m-введенных n-угольников имеет наибольшую площадь.

**Результат работы программы:**

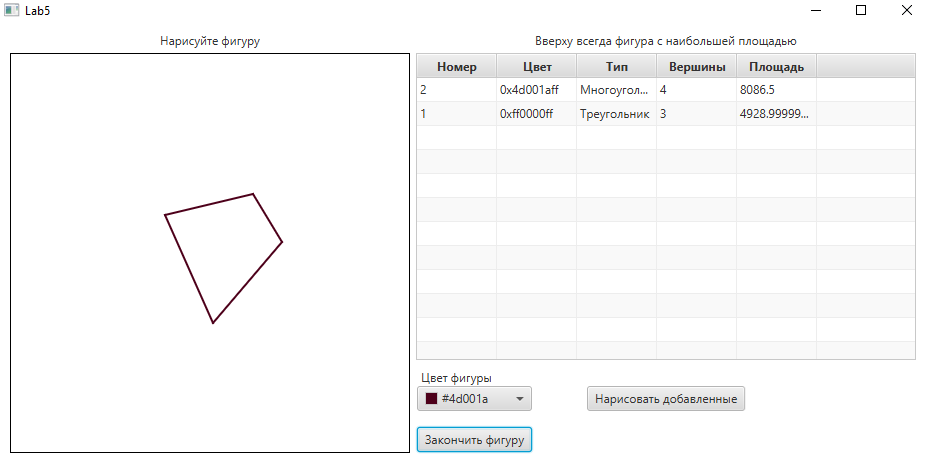
****

Рисунок 1 – Окно программы после добавления треугольника и многоугольника

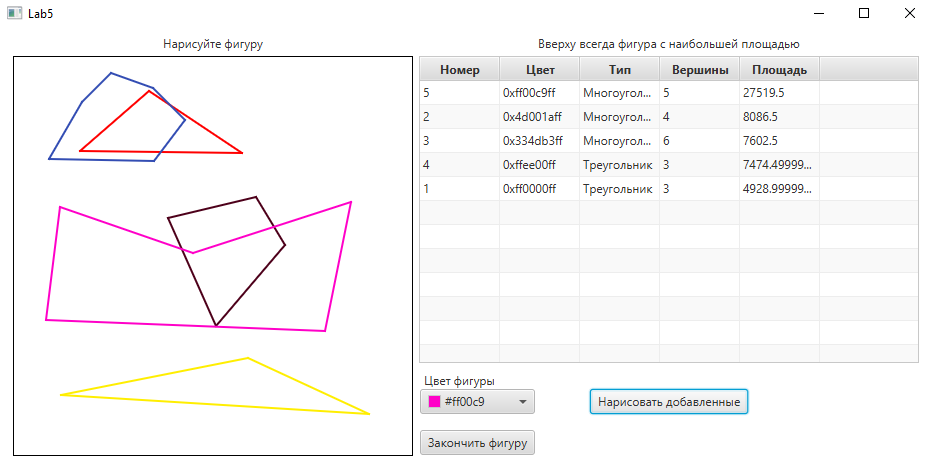


Рисунок 2 – Окно программы после добавления нескольких фигур и вывода их всех на экран для сравнения

**Вывод:** в результате выполнения лабораторной работы была изучена работа с графическим интерфейсом (JavaFX) в языке программирования *Java,* написана программа, использующая графический интерфейс.

**Приложение А**

**Исходный код программы**

**Main.java:**

**public class** Main **extends** Application {  
  
 @Override  
 **public void** start(Stage primaryStage) **throws** Exception{  
 MainController mainController = **new** MainController();  
 FXMLLoader loader = **new** FXMLLoader(getClass().getResource(**"main.fxml"**));  
 loader.setController(mainController);  
  
 Scene scene = **new** Scene(loader.load());  
 primaryStage.setTitle(**"Lab5"**);  
 primaryStage.setScene(scene);  
 primaryStage.show();  
  
 mainController.setValues();  
 }  
  
 **public static void** main(String[] args) {  
 *launch*(args);  
 }  
}

**MainController.java:**

**class** MainController {  
  
 @FXML **private** Canvas **canvas**;  
 @FXML **private** ColorPicker **colorPicker**;  
 @FXML **private** TableView<FigureContainer> **tableView**;  
  
 **private** GraphicsContext **context**;  
 **private boolean begin** = **false**;  
 **private** List<Point> **points** = **new** ArrayList<>();  
  
 **private** List<FigureContainer> **figureContainers** = **new** ArrayList<>();  
  
 **public void** setValues() {  
 **context** = **canvas**.getGraphicsContext2D();  
 **context**.setLineWidth(2);  
 **colorPicker**.setValue(Color.***RED***);  
  
 createTable();  
 }  
  
 **private void** createTable() {  
 **tableView**.getColumns().clear();  
 TableColumn<FigureContainer, Integer> idColumn = **new** TableColumn<>(**"Номер"**);  
 TableColumn<FigureContainer, Color> colorColumn = **new** TableColumn<>(**"Цвет"**);  
 TableColumn<FigureContainer, String> typeColumn = **new** TableColumn<>(**"Тип"**);  
 TableColumn<FigureContainer, Integer> anglesColumn = **new** TableColumn<>(**"Вершины"**);  
 TableColumn<FigureContainer, Double> areaColumn = **new** TableColumn<>(**"Площадь"**);  
  
 idColumn.setCellValueFactory(**new** PropertyValueFactory<>(**"id"**));  
 colorColumn.setCellValueFactory(**new** PropertyValueFactory<>(**"color"**));  
 typeColumn.setCellValueFactory(**new** PropertyValueFactory<>(**"type"**));  
 anglesColumn.setCellValueFactory(**new** PropertyValueFactory<>(**"angles"**));  
 areaColumn.setCellValueFactory(**new** PropertyValueFactory<>(**"area"**));  
  
 idColumn.setSortable(**false**);  
 colorColumn.setSortable(**false**);  
 typeColumn.setSortable(**false**);  
 anglesColumn.setSortable(**false**);  
 areaColumn.setSortable(**false**);  
  
 **tableView**.getColumns().addAll(idColumn, colorColumn, typeColumn, anglesColumn, areaColumn);  
 }  
  
 @FXML  
 **private void** clickOnCanvas(MouseEvent e) {  
 **if** (**begin**) {  
 **context**.strokeLine(**points**.get(**points**.size() - 1).getX(),  
 **points**.get(**points**.size() - 1).getY(), e.getX(), e.getY());  
 **points**.add(**new** Point(e.getX(), e.getY()));  
 } **else** {  
 **context**.clearRect(0, 0, **canvas**.getWidth(), **canvas**.getHeight());  
 **begin** = **true**;  
 **context**.beginPath();  
 **context**.setStroke(**colorPicker**.getValue());  
 **context**.moveTo(e.getX(), e.getY());  
 **points**.add(**new** Point(e.getX(), e.getY()));  
 }  
 }  
  
 **private void** reset() {  
 **context**.clearRect(0, 0, **canvas**.getWidth(), **canvas**.getHeight());  
 **points** = **new** ArrayList<>();  
 **context**.closePath();  
 **begin** = **false**;  
 }  
  
 @FXML  
 **private void** endFigure() {  
 **if** (**points**.size() < 3) {  
 Alert alert = **new** Alert(Alert.AlertType.***ERROR***);  
 alert.setTitle(**"Ошибка"**);  
 alert.setHeaderText(**"Количество вершин должно быть не менее трёх!"**);  
 alert.showAndWait();  
 reset();  
 **return**;  
 }  
 **context**.strokeLine(**points**.get(0).getX(), **points**.get(0).getY(),  
 **points**.get(**points**.size() - 1).getX(), **points**.get(**points**.size() - 1).getY());  
 **context**.closePath();  
 **begin** = **false**;  
  
 Figure figure;  
 **try** {  
 **if** (**points**.size() == 3) {  
 figure = **new** Triangle(**new** Point(**points**.get(0).getX(), **points**.get(0).getY()),  
 **new** Point(**points**.get(1).getX(), **points**.get(1).getY()),  
 **new** Point(**points**.get(2).getX(), **points**.get(2).getY()));  
 } **else** {  
 Point[] pointArray = **new** Point[1];  
 pointArray = **points**.toArray(pointArray);  
 figure = **new** PolyAngle(pointArray);  
 }  
 } **catch** (IllegalArgumentException e) {  
 Alert alert = **new** Alert(Alert.AlertType.***ERROR***);  
 alert.setTitle(**"Ошибка"**);  
 alert.setHeaderText(**"Линии между вершинами фигуры не должны "** +  
 **"пересекаться и/или многоугольник должен быть выпуклым!"**);  
 alert.showAndWait();  
 reset();  
 **return**;  
 }  
 **points** = **new** ArrayList<>();  
 **figureContainers**.add(**new** FigureContainer(figure, **figureContainers**.size() + 1, (Color) **context**.getStroke()));  
 **tableView**.setItems(**new** SortedList<>(FXCollections.*observableList*(**figureContainers**),  
 (o1, o2) -> Double.*compare*(o2.getArea(), o1.getArea())));  
 }  
  
 @FXML **private void** drawFigures() {  
 reset();  
 **for** (FigureContainer figureContainer : **figureContainers**) {  
 Point[] pointArray = figureContainer.getPoints();  
 **context**.beginPath();  
 **context**.setStroke(figureContainer.**color**);  
 **context**.moveTo(pointArray[0].getX(), pointArray[0].getY());  
 **for** (**int** i = 1; i < pointArray.**length**; i++) {  
 **context**.strokeLine(pointArray[i - 1].getX(),  
 pointArray[i - 1].getY(),  
 pointArray[i].getX(),  
 pointArray[i].getY());  
 }  
 **context**.strokeLine(pointArray[0].getX(), pointArray[0].getY(),  
 pointArray[pointArray.**length** - 1].getX(),  
 pointArray[pointArray.**length** - 1].getY());  
 **context**.closePath();  
 }  
 }  
  
  
 **public static class** FigureContainer {  
 **private** Figure **figure**;  
 **private int id**;  
 **private** Color **color**;  
  
 FigureContainer(Figure figure, **int** id, Color color) {  
 **this**.**figure** = figure;  
 **this**.**id** = id;  
 **this**.**color** = color;  
 }  
  
 **public int** getId() {  
 **return id**;  
 }  
  
 **public** Color getColor() {  
 **return color**;  
 }  
  
 **public** String getType() {  
 **return figure**.getName();  
 }  
  
 **public int** getAngles() {  
 **return figure**.getPoints().**length**;  
 }  
  
 **double** getArea() {  
 **return figure**.getArea();  
 }  
  
 Point[] getPoints() {  
 **return figure**.getPoints();  
 }  
 }  
}

**Point.kt:**

**class** Point(**var x**: Double, **var y**: Double) {  
  
 **fun** distanceTo(another: Point): Double {  
 **return** *sqrt*((**x** - another.**x**).*pow*(2.0) + (**y** - another.**y**).*pow*(2.0))  
 }  
}

**Figure.kt:**

**abstract class** Figure {  
 **abstract fun** getName(): String  
 **abstract fun** getArea(): Double  
 **abstract fun** getPoints(): Array<Point>  
}

**Triangle.java:**

**public class** Triangle **extends** Figure {  
  
 **private** Point **a**, **b**, **c**;  
 **private double line1**, **line2**, **line3**;  
  
 **public** Triangle(Point a, Point b, Point c) **throws** IllegalArgumentException {  
  
 **line1** = a.distanceTo(b);  
 **line2** = a.distanceTo(c);  
 **line3** = b.distanceTo(c);  
  
 **if** (((**line1** + **line2**) <= **line3**) || ((**line1** + **line3**) <= **line2**) || ((**line2** + **line3**) <= **line1**)) {  
 **throw new** IllegalArgumentException(**"The triangle with the same coordinates may not exist"**);  
 }  
  
 **this**.**a** = a;  
 **this**.**b** = b;  
 **this**.**c** = c;  
 }  
  
 @Override  
 **public** String getName() {  
 **return "Треугольник"**;  
 }  
  
 @Override  
 **public double** getArea() {  
 **double** p = (**line1** + **line2** + **line3**) / 2;  
 **return** Math.*sqrt*(p \* (p - **line1**) \* (p - **line2**) \* (p - **line3**));  
 }  
  
 @Override  
 **public** Point[] getPoints() {  
 **return new** Point[]{**a**, **b**, **c**};  
 }  
}

**PolyAngle.java:**

**public class** PolyAngle **extends** Figure {  
  
 **private** Point[] **points**;  
 **private double**[] **lines**;  
  
 **public** PolyAngle(Point... points) **throws** IllegalArgumentException {  
 **if** (points.**length** <= 3) {  
 **throw new** IllegalArgumentException(**"A polygon must consist of 4 or more vertices"**);  
 }  
  
 checkForConvex(points);  
  
 **double**[] temp = **new double**[points.**length**];  
 **for** (**int** i = 1; i < points.**length**; i++) {  
 temp[i] = points[i].distanceTo(points[i - 1]);  
 }  
 temp[0] = points[0].distanceTo(points[points.**length** - 1]);  
  
 **this**.**points** = points;  
 **this**.**lines** = temp;  
 }  
  
 **private void** checkForConvex(Point[] points) **throws** IllegalArgumentException {  
 **double** first = checkPoints(points[points.**length** - 1], points[0], points[1]);  
 **for** (**int** i = 1; i < points.**length** - 1; i++) {  
 **if** (Double.*compare*(Math.*signum*(checkPoints(points[i - 1], points[i], points[i + 1])), Math.*signum*(first)) != 0) {  
 **throw new** IllegalArgumentException(**"The polygon must be convex"**);  
 }  
 }  
 }  
  
 **private double** checkPoints(Point a, Point b, Point c) {  
 Point ab = **new** Point(b.getX() - a.getX(),  
 b.getY() - a.getY());  
 Point bc = **new** Point(c.getX() - b.getX(),  
 c.getY() - b.getY());  
 **return** ab.getX() \* bc.getY() - ab.getY() \* bc.getX();  
 }  
  
 @Override  
 **public** String getName() {  
 **return "Многоугольник"**;  
 }  
  
 @Override  
 **public double** getArea() {  
 **double** area = 0;  
 **for** (**int** i = 0; i < **points**.**length** - 1; i++) {  
 area += (**points**[i].getX() + **points**[i + 1].getX())  
 \* (**points**[i].getY() - **points**[i + 1].getY());  
 }  
 area += (**points**[**points**.**length** - 1].getX() + **points**[0].getX())  
 \* (**points**[**points**.**length** - 1].getY() - **points**[0].getY());  
 **return** 0.5 \* Math.*abs*(area);  
 }  
  
 @Override  
 **public** Point[] getPoints() {  
 **return points**;  
 }  
}