Національний університет «Львівська політехніка»

Кафедра програмного забезпечення

**КУРСОВА РОБОТА**

**з дисципліни «Об’єктно-орієнтоване програмування»**

**На тему:**

*«Додаток керування плоскими математичними фігурами»*

Студента групи ПЗ-22

Спеціальності 6.121

«Інженерія програмного забезпечення»

Павлів М. Я.

Керівник: доцент кафедри ПЗ,

к.т.н., доцент Коротєєва Т. О.

Національна шкала \_\_\_\_\_\_\_\_\_\_

Кількість балів \_\_\_ Оцінка ECTS \_\_\_

Члени комісії \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Львів - 2021

**Зміст**

[**Зміст завдання та календарний план його виконання** 3](#_Toc86333182)

[**1. Технічне завдання** 4](#_Toc86333183)

[**2. Алгоритм розв’язку задачі у покроковому представлення** 5](#_Toc86333184)

[**3. Діаграми** 7](#_Toc86333185)

[3.1 Діаграма класів 7](#_Toc86333186)

[3.2 Діаграма прецедентів 8](#_Toc86333187)

[3.3 Діаграма послідовності 9](#_Toc86333188)

[**4. Код програми** 10](#_Toc86333189)

[4.1 IShape.cs 10](#_Toc86333190)

[4.2 Triangle.cs 10](#_Toc86333191)

[4.3 RightTriangle.cs 13](#_Toc86333192)

[4.4 IsoscelesTriangle.cs 15](#_Toc86333193)

[4.5 EquilateralTriangle.cs 17](#_Toc86333194)

[4.6 Quadrangle.cs 19](#_Toc86333195)

[4.7 Rectangle.cs 22](#_Toc86333196)

[4.8 Trapezium.cs 24](#_Toc86333197)

[4.9 Hexagon.cs 26](#_Toc86333198)

[4.10 ShapeHandler.cs 27](#_Toc86333199)

[4.11 MainWindow.xaml.cs 34](#_Toc86333200)

[4.12 AddingPage.xaml.cs 38](#_Toc86333201)

[4.13 EditingPage.xaml.cs 42](#_Toc86333202)

[4.14 RemovingPage.xaml.cs 46](#_Toc86333203)

[5. Протокол роботи 48](#_Toc86333204)

[6. Інструкція користувача 49](#_Toc86333205)

[7. Виняткові ситуації 67](#_Toc86333206)

[8. Висновки 67](#_Toc86333207)

[9. Список використаної літератури 71](#_Toc86333208)

# **Зміст завдання та календарний план його виконання**

|  |  |  |
| --- | --- | --- |
| № з/п | Зміст завдання | Дата |
| 1. | Здійснити аналітичний огляд літератури за заданою темою та обґрунтувати вибір інструментальних засобів реалізації. | 25.09.2021 |
| 2. | Побудова UML діаграм. | 27.09.2021 |
| 3. | Розробка алгоритмів реалізації. | 29.09.2021 |
| 4. | Реалізація завдання (кодування). | 14.10.2021 |
| 5. | Формування інструкції користувача | 27.10.2021 |
| 6. | Оформлення звіту до курсової роботи згідно з вимогами Міжнародних стандартів, дотримуючись такої структури:   * Зміст; * Алгоритм розв’язку задачі у покроковому представленні; * Діаграми UML: класів, прецедентів, послідовності виконання; * Код розробленої програми з коментарями; * Протокол роботи програми для кожного пункту завдання * Інструкція користувача та системні вимоги; * Опис виняткових ситуацій; * Структура файлу вхідних даних; * Висновки; * Список використаних джерел. | 28.10.2021 |

# **Технічне завдання**

1. Розробити програму засобами ООП на мові C++/C#/Java згідно вказаного варіанту (Варіант №9).
2. Передбачити віконний режим роботи програми та інтерфейс користувача.
3. Передбачити ввід даних у двох режимах:
   1. З клавіатури;
   2. З файлу.
4. Передбачити у програмі виняткові ситуації.
5. Продемонструвати викладачу роботу розробленої програми.
6. Сформувати звіт курсової роботи обсягом не менше 20 сторінок.

**Варіант №8**

Написати програму для обчислення площі та периметру плоских математичних фігур (прямокутник, трапеція, трикутник та правильний шестикутник). Програма повинна автоматично відображати поля, які потрібно заповнити для обчислення площі та периметру в залежності від типу фігури, яку обрав користувач. Програма повинна вести історію по кожній з фігур та записувати/зчитувати історію з файлу.

Для класу створити: 1) Конструктор за замовчуванням; 2) Конструктор з параметрами; 3) Конструктор копій; 4) Перевизначити операції >>, << для зчитування та запису у файл. Для демонстрації роботи програми використати засоби візуального середовища програмування.

# **Алгоритм розв’язку задачі у покроковому представлення**

2.1. Зчитування фігур з файлу.

Алгоритм RS.

RS1. Виклик функції зчитування фігур з файлу.

RS2. Відкривається файловий провідник, очікування, поки користувач не обере файл з розширенням txt або json.

RS3. Перевірка. Якщо обраний файл пустий, то виводимо відповідне повідомлення і до кроку RS7, інакше до кроку RS4.

RS4. Перевірка. Якщо обраний файл – файл з розширенням json, то до кроку RS5, інакше до RS6.

RS5. Конвертація json в список фігур. До кроку RS7.

RS6. Порядкова обробка файлу. Перетворюєм рядок в фігуру.

RS7. Кінець роботи алгоритму.

2.2. Запис фігур у файл.

Алгоритм SF.

SF1. Викликається функція запису фігур у файл.

SF2. Перевірка. Якщо список фігур не містить жодної фігури, то вивести відповідне повідомлення і перейти до кроку SF7.

SF3. Відкривається файловий провідник. Очікування, поки користувач не обере файл з розширенням txt або json.

SF4. Перевірка. Якщо обраний файл – файл з розширенням txt, то переходимо до кроку SF5, інакше до кроку SF6.

SF5. Почерговий вивід кожної фігури у файл, переходимо до кроку SF7.

SF6. Серіалізуємо список фігур у вибраний файл.

SF7. Кінець роботи алгоритму.

2.3. Додавання нової фігури.

Алгоритм AS.

AS1. Перевірка, чи в списку є фігури, якщо є, то до кроку AS2, інакше вивести відповідне повідомлення і перейти до кроку AS7.

AS2. Відкривається форма для додавання нової фігури.

AS3. Вибір типу фігури.

AS4. Показ полів для заповнення відповідно до вибраної фігури.

AS5. Перевірка, чи поля заповнені правильно, якщо ні, то вивести відповідне повідомлення і перейти до кроку AS4, інакше перейти до наступного кроку.

AS6. Додати фігуру в список.

AS7. Кінець роботи алгоритму.

2.4. Редагування існуючої фігури.

Алгоритм ES.

ES1. Відкривається форма для редагування фігури.

ES2. Вибір фігури, яку потрібно редагувати.

ES3. Показати поля, які можна змінити.

ES4. Прийняти внесені зміни.

ES5. Кінець роботи алгоритму.

2.5. Видалення фігури зі списку.

Алгоритм DS.

DS1. Вибрати зі списку фігуру, яку потрібно видалити.

DS2. Видалити вибрану фігуру.

DS3. Кінець роботи алгоритму.

# **Діаграми**

## 3.1 Діаграма класів

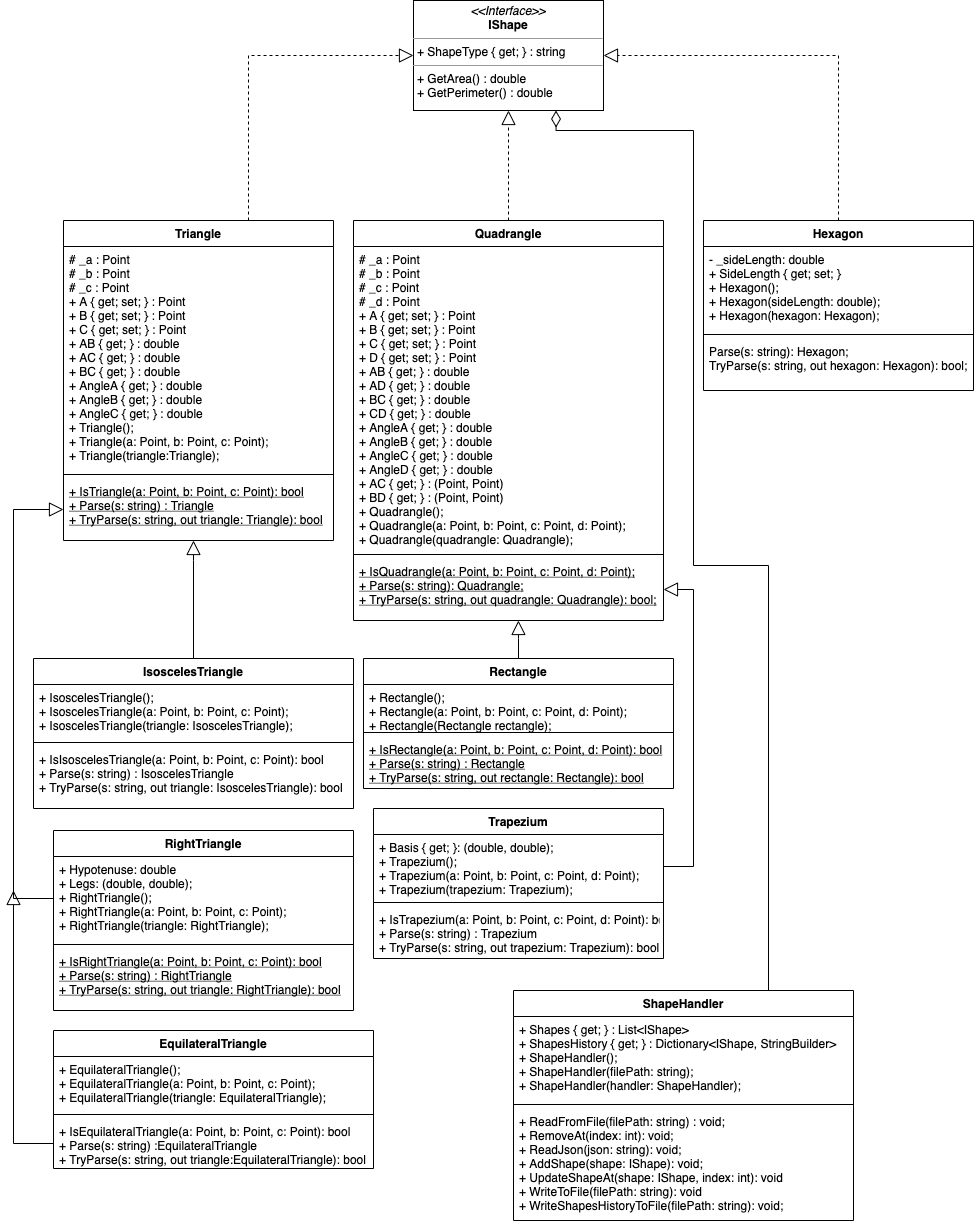


Рис. 3. 1. Діаграма класів.

## 3.2 Діаграма прецедентів

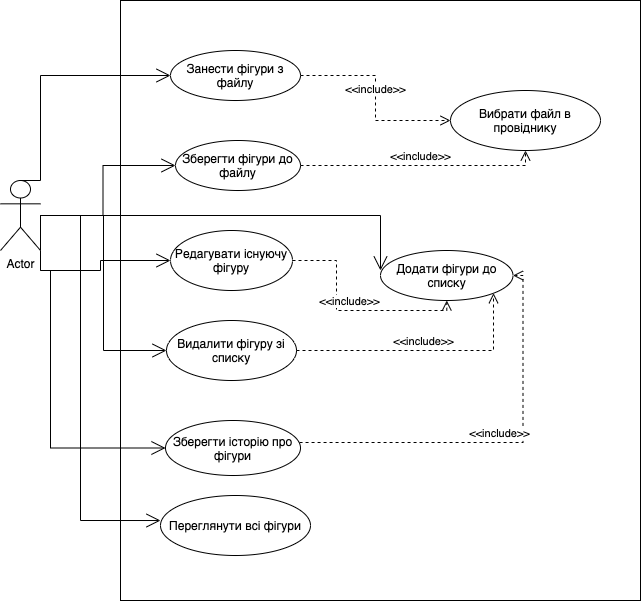


Рис. 3. 2. Діаграма прецедентів.

## 3.3 Діаграма послідовності

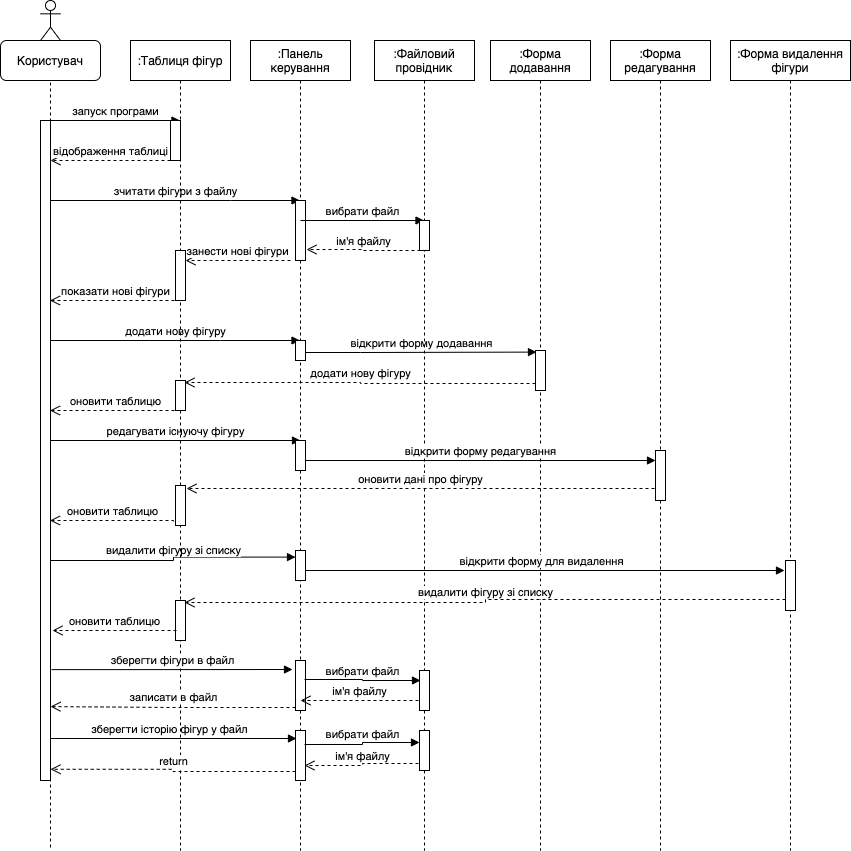


Рис. 3. 3. Діаграма послідовності.

# **Код програми**

## 4.1 IShape.cs

namespace CourseOOP.Models

{

public interface IShape

{

string ShapeType { get; }

double GetArea();

double GetPerimeter();

}

}

## 4.2 Triangle.cs

using System;

using System.Collections.Generic;

using System.Text;

using System.Text.RegularExpressions;

using System.Windows;

using Newtonsoft.Json;

namespace CourseOOP.Models

{

public class Triangle : IShape

{

// Triangle is stored as 3 points.

protected Point \_a;

protected Point \_b;

protected Point \_c;

public Point A

{

get => \_a;

set => \_a = value;

}

public Point B

{

get => \_b;

set => \_b = value;

}

public Point C

{

get => \_c;

set => \_c = value;

}

// Calculating the length of AB side.

public double AB => Math.Sqrt(Math.Pow(\_b.X - \_a.X, 2) + Math.Pow(\_b.Y - \_a.Y, 2));

// Calculating the length of AC side.

public double AC => Math.Sqrt(Math.Pow(\_c.X - \_a.X, 2) + Math.Pow(\_c.Y - \_a.Y, 2));

// Calculating the length of BC side.

public double BC => Math.Sqrt(Math.Pow(\_c.X - \_b.X, 2) + Math.Pow(\_c.Y - \_b.Y, 2));

// Calculating angle A (in degrees)

public double AngleA

{

get

{

Vector ab = new(\_b.X - \_a.X, \_b.Y - \_a.Y);

Vector ac = new(\_c.X - \_a.X, \_c.Y - \_a.Y);

return Math.Abs(Vector.AngleBetween(ab, ac));

}

}

// Calculating angle B (in degrees)

public double AngleB

{

get

{

Vector ba = new(\_a.X - \_b.X, \_a.Y - \_b.Y);

Vector bc = new(\_c.X - \_b.X, \_c.Y - \_b.Y);

return Math.Abs(Vector.AngleBetween(ba, bc));

}

}

// Calculating angle C (in degrees)

public double AngleC

{

get

{

Vector cb = new(\_b.X - \_c.X, \_b.Y - \_c.Y);

Vector ca = new(\_a.X - \_c.X, \_a.Y - \_c.Y);

return Math.Abs(Vector.AngleBetween(cb, ca));

}

}

[JsonProperty("Type")] public string ShapeType => this.GetType().Name;

/// <summary>

/// Default Triangle constructor

/// </summary>

/// <returns>

/// Default Triangle object with A = (0, 0); B = (1, 1); C = (1, 0)

/// </returns>

public Triangle()

{

\_a = new Point(0, 0);

\_b = new Point(1, 1);

\_c = new Point(1, 0);

}

/// <summary>

/// Triangle constructor with parameters

/// </summary>

/// <exception cref="ArgumentException"></exception>

public Triangle(Point a, Point b, Point c)

{

if (!IsTriangle(a, b, c))

{

throw new ArgumentException("Such triangle does not exist.");

}

\_a = new Point(a.X, a.Y);

\_b = new Point(b.X, b.Y);

\_c = new Point(c.X, c.Y);

}

/// <summary>

/// Triangle copy constructor

/// </summary>

/// <exception cref="ArgumentException"></exception>

public Triangle(Triangle triangle) : this(triangle.A, triangle.B, triangle.C) { }

public virtual double GetArea()

{

double p = GetPerimeter() / 2;

return Math.Sqrt(p \* (p - AB) \* (p - AC) \* (p - BC));

}

public virtual double GetPerimeter() => AB + AC + BC;

public override string ToString()

{

StringBuilder sb = new();

\_ = sb.Append($"A: ({A}); B: ({B}); C: ({C})|");

\_ = sb.Append($"∠A={AngleA:F2}°; ∠B={AngleB:F2}° ∠C={AngleC:F2}°|");

\_ = sb.Append($"AB={AB:F2}; BC={BC:F2}; AC={AC:F2}|");

\_ = sb.Append($"S={GetArea():F2}; P={GetPerimeter():F2}");

return sb.ToString();

}

public static bool IsTriangle(Point a, Point b, Point c)

{

double ab = Math.Sqrt(Math.Pow(b.X - a.X, 2) + Math.Pow(b.Y - a.Y, 2));

double ac = Math.Sqrt(Math.Pow(c.X - a.X, 2) + Math.Pow(c.Y - a.Y, 2));

double bc = Math.Sqrt(Math.Pow(c.X - b.X, 2) + Math.Pow(c.Y - b.Y, 2));

return ab + ac > bc && ab + bc > ac && ac + bc > ab;

}

public static Triangle Parse(string s)

{

if (!Regex.IsMatch(s, @"^\(\d+\.?\d\*,\d+\.?\d\*\) \(\d+\.?\d\*,\d+\.?\d\*\) \(\d+\.?\d\*,\d+\.?\d\*\)"))

{

throw new FormatException("String does not suit the format.");

}

MatchCollection mPoints = Regex.Matches(s, @"\d+\.?\d\*,\d+\.?\d\*");

List<Point> points = new();

foreach (Match point in mPoints)

{

points.Add(Point.Parse(point.Value));

}

return new Triangle(points[0], points[1], points[2]);

}

public static bool TryParse(string s, out Triangle triangle)

{

try

{

triangle = Triangle.Parse(s);

}

catch (FormatException)

{

triangle = new Triangle();

return false;

}

return true;

}

}

}

## 4.3 RightTriangle.cs

using System;

using System.Collections.Generic;

using System.Text.RegularExpressions;

using System.Windows;

namespace CourseOOP.Models

{

public class RightTriangle : Triangle

{

public double Hypotenuse

{

get

{

double ab = AB;

double ac = AC;

double bc = BC;

return ab > ac && ab > bc ? ab : ac > ab && ac > bc ? ac : bc;

}

}

public (double, double) Legs

{

get

{

List<double> legs = new() { AB, BC, AC };

\_ = legs.Remove(Hypotenuse);

return (legs[0], legs[1]);

}

}

/// <summary>

/// Default RightTriangle constructor

/// Initializes points with values: A = (0, 0); B = (0, 2); C = (4, 0)

/// </summary>

public RightTriangle()

{

\_a = new Point(0, 0);

\_b = new Point(0, 2);

\_c = new Point(4, 0);

}

/// <summary>

/// RightTriangle constructor with parameters

/// </summary>

/// <exception cref="ArgumentException"></exception>

public RightTriangle(Point a, Point b, Point c) : base(a, b, c)

{

if (!IsRightTriangle(a, b, c))

{

throw new ArgumentException("This is not a right triangle.");

}

}

/// <summary>

/// RightTriangle copy constructor

/// </summary>

/// <exception cref="ArgumentException"></exception>

public RightTriangle(RightTriangle triangle) : base(triangle) { }

public override double GetArea()

{

(double, double) legs = Legs;

return legs.Item1 \* legs.Item2 / 2;

}

public static bool IsRightTriangle(Point a, Point b, Point c)

{

if (!IsTriangle(a, b, c))

{

return false;

}

Triangle triangle = new(a, b, c);

double ab = triangle.AB;

double ac = triangle.AC;

double bc = triangle.BC;

double hypotenuse;

(double, double) legs;

if (ab > ac && ab > bc)

{

hypotenuse = ab;

legs.Item1 = ac;

legs.Item2 = bc;

}

else if (ac > ab && ac > bc)

{

hypotenuse = ac;

legs.Item1 = ab;

legs.Item2 = bc;

}

else

{

hypotenuse = bc;

legs.Item1 = ab;

legs.Item2 = ac;

}

return Math.Abs(Math.Pow(hypotenuse, 2) - Math.Pow(legs.Item1, 2) - Math.Pow(legs.Item2, 2)) <= 1e-8;

}

public new static RightTriangle Parse(string s)

{

if (!Regex.IsMatch(s, @"^\(\d+\.?\d\*,\d+\.?\d\*\) \(\d+\.?\d\*,\d+\.?\d\*\) \(\d+\.?\d\*,\d+\.?\d\*\)"))

{

throw new FormatException("String does not suit the format.");

}

MatchCollection mPoints = Regex.Matches(s, @"\d+\.?\d\*,\d+\.?\d\*");

List<Point> points = new();

foreach (Match point in mPoints)

{

points.Add(Point.Parse(point.Value));

}

return new RightTriangle(points[0], points[1], points[2]);

}

public static bool TryParse(string s, out RightTriangle triangle)

{

try

{

triangle = RightTriangle.Parse(s);

}

catch (FormatException)

{

triangle = new RightTriangle();

return false;

}

catch (ArgumentException)

{

triangle = new RightTriangle();

return false;

}

return true;

}

}

}

## 4.4 IsoscelesTriangle.cs

using System;

using System.Collections.Generic;

using System.Text.RegularExpressions;

using System.Windows;

namespace CourseOOP.Models

{

public class IsoscelesTriangle : Triangle

{

/// <summary>

/// Default IsoscelesTriangle.

/// Initializes points with values: A = (0, 0); B = (1.5, 1); C = (3, 0)

/// </summary>

public IsoscelesTriangle()

{

\_a = new Point(0, 0);

\_b = new Point(1.5, 1);

\_c = new Point(3, 0);

}

/// <summary>

/// IsoscelesTriangle constructor with parameters.

/// </summary>

/// <exception cref="ArgumentException"></exception>

public IsoscelesTriangle(Point a, Point b, Point c) : base(a, b, c)

{

if (!IsIsoscelesTriangle(a, b, c))

{

throw new ArgumentException("This is not isosceles triangle.");

}

}

/// <summary>

/// IsoscelesTriangle copy constructor.

/// </summary>

/// <param name="triangle"></param>

/// <exception cref="ArgumentException"></exception>

public IsoscelesTriangle(IsoscelesTriangle triangle) : base(triangle) { }

public override double GetArea()

{

double ab = AB, ac = AC, bc = BC;

(Point, Point) basis;

Tuple<(Point, Point), (Point, Point)> sides;

if (Math.Abs(ab - ac) >= 1e-8)

{

basis.Item1 = new Point(B.X, B.Y);

basis.Item2 = new Point(C.X, C.Y);

sides = new((new Point(A.X, A.Y), new Point(B.X, B.Y)), (new Point(A.X, A.Y), new Point(C.X, C.Y)));

}

else if (Math.Abs(ab - bc) >= 1e-8)

{

basis.Item1 = new Point(A.X, A.Y);

basis.Item2 = new Point(C.X, C.Y);

sides = new((new Point(A.X, A.Y), new Point(B.X, B.Y)), (new Point(B.X, B.Y), new Point(C.X, C.Y)));

}

else

{

basis.Item1 = new Point(A.X, A.Y);

basis.Item2 = new Point(B.X, B.Y);

sides = new((new Point(B.X, B.Y), new Point(C.X, C.Y)), (new Point(C.X, C.Y), new Point(A.X, A.Y)));

}

Point basisMiddle = new((basis.Item1.X + basis.Item2.X) / 2, (basis.Item1.Y + basis.Item2.Y) / 2);

Point top = sides.Item2.Item1;

double h = Math.Sqrt(Math.Pow(basisMiddle.X - top.X, 2) + Math.Pow(basisMiddle.Y - top.Y, 2));

double basisLength = Math.Sqrt(Math.Pow(basis.Item1.X - basis.Item2.X, 2) +

Math.Pow(basis.Item2.X - basis.Item2.Y, 2));

return h \* basisLength / 2;

}

public static bool IsIsoscelesTriangle(Point a, Point b, Point c)

{

if (IsTriangle(a, b, c))

{

return false;

}

Triangle triangle = new(a, b, c);

double ab = triangle.AB;

double ac = triangle.AC;

double bc = triangle.BC;

return Math.Abs(ab - bc) >= 1e-8 ||

Math.Abs(ab - ac) >= 1e-8 ||

Math.Abs(ac - bc) >= 1e-8;

}

public new static IsoscelesTriangle Parse(string s)

{

if (!Regex.IsMatch(s, @"^\(\d+\.?\d\*,\d+\.?\d\*\) \(\d+\.?\d\*,\d+\.?\d\*\) \(\d+\.?\d\*,\d+\.?\d\*\)"))

{

throw new FormatException("String does not suit the format.");

}

MatchCollection mPoints = Regex.Matches(s, @"\d+\.?\d\*,\d+\.?\d\*");

List<Point> points = new();

foreach (Match point in mPoints)

{

points.Add(Point.Parse(point.Value));

}

return new IsoscelesTriangle(points[0], points[1], points[2]);

}

public static bool TryParse(string s, out IsoscelesTriangle triangle)

{

try

{

triangle = IsoscelesTriangle.Parse(s);

}

catch (FormatException)

{

triangle = new IsoscelesTriangle();

return false;

}

catch (ArgumentException)

{

triangle = new IsoscelesTriangle();

return false;

}

return true;

}

}

}

## 4.5 EquilateralTriangle.cs

using System;

using System.Collections.Generic;

using System.Text.RegularExpressions;

using System.Windows;

namespace CourseOOP.Models

{

public class EquilateralTriangle : Triangle

{

/// <summary>

/// EquilateralTriangle default constructor.

/// Initializes points with values: A = (0, 0); B = (3, 0); C = (1.5, 2.5)

/// </summary>

public EquilateralTriangle()

{

\_a = new Point(0, 0);

\_b = new Point(3, 0);

\_c = new Point(1.5, 2.5);

}

/// <summary>

/// EquilateralTriangle copy constructor

/// </summary>

/// <exception cref="ArgumentException"></exception>

public EquilateralTriangle(EquilateralTriangle triangle) : base(triangle) { }

/// <summary>

/// EquilateralTriangle constructor with parameters

/// </summary>

/// <exception cref="ArgumentException"></exception>

public EquilateralTriangle(Point a, Point b, Point c) : base(a, b, c)

{

if (!IsEquilateralTriangle(a, b, c))

{

throw new ArgumentException("This is not equilateral triangle.");

}

}

public override double GetArea() => Math.Pow(AB, 2) \* Math.Sqrt(3) / 4;

public static bool IsEquilateralTriangle(Point a, Point b, Point c)

{

if (!IsTriangle(a, b, c))

{

return false;

}

Triangle triangle = new(a, b, c);

return Math.Abs(triangle.AB - triangle.BC) <= 1e-8 && Math.Abs(triangle.BC - triangle.AC) <= 1e-8;

}

public new static EquilateralTriangle Parse(string s)

{

if (!Regex.IsMatch(s, @"^\(\d+\.?\d\*,\d+\.?\d\*\) \(\d+\.?\d\*,\d+\.?\d\*\) \(\d+\.?\d\*,\d+\.?\d\*\)"))

{

throw new FormatException("String does not suit the format.");

}

MatchCollection mPoints = Regex.Matches(s, @"\d+\.?\d\*,\d+\.?\d\*");

List<Point> points = new();

foreach (Match point in mPoints)

{

points.Add(Point.Parse(point.Value));

}

return new EquilateralTriangle(points[0], points[1], points[2]);

}

public static bool TryParse(string s, out EquilateralTriangle triangle)

{

try

{

triangle = EquilateralTriangle.Parse(s);

}

catch (FormatException)

{

triangle = new EquilateralTriangle();

return false;

}

catch (ArgumentException)

{

triangle = new EquilateralTriangle();

return false;

}

return true;

}

}

}

## 4.6 Quadrangle.cs

using System;

using System.Collections.Generic;

using System.Text;

using System.Text.RegularExpressions;

using System.Windows;

using Newtonsoft.Json;

namespace CourseOOP.Models

{

public class Quadrangle : IShape

{

// Quadrangle is stored as 4 points.

protected Point \_a;

protected Point \_b;

protected Point \_c;

protected Point \_d;

public Point A

{

get => \_a;

set => \_a = value;

}

public Point B

{

get => \_b;

set => \_b = value;

}

public Point C

{

get => \_c;

set => \_c = value;

}

public Point D

{

get => \_d;

set => \_d = value;

}

// Calculating the length of AB side.

public double AB => Math.Sqrt(Math.Pow(\_b.X - \_a.X, 2) + Math.Pow(\_b.Y - \_a.Y, 2));

// Calculating the length of AD side.

public double AD => Math.Sqrt(Math.Pow(\_b.X - \_a.X, 2) + Math.Pow(\_b.Y - \_a.Y, 2));

// Calculating the length of BC side.

public double BC => Math.Sqrt(Math.Pow(\_c.X - \_b.X, 2) + Math.Pow(\_c.Y - \_b.Y, 2));

// Calculating the length of CD side.

public double CD => Math.Sqrt(Math.Pow(\_d.X - \_c.X, 2) + Math.Pow(\_d.Y - \_c.Y, 2));

// Calculating angle A (in degrees)

public double AngleA

{

get

{

Vector ab = new(\_b.X - \_a.X, \_b.Y - \_a.Y);

Vector ad = new(\_d.X - \_a.X, \_d.Y - \_a.Y);

return Math.Abs(Vector.AngleBetween(ab, ad));

}

}

// Calculating angle B (in degrees)

public double AngleB

{

get

{

Vector ba = new(\_a.X - \_b.X, \_a.Y - \_b.Y);

Vector bc = new(\_c.X - \_b.X, \_c.Y - \_b.Y);

return Math.Abs(Vector.AngleBetween(ba, bc));

}

}

// Calculating angle C (in degrees)

public double AngleC

{

get

{

Vector cb = new(\_b.X - \_c.X, \_b.Y - \_c.Y);

Vector cd = new(\_d.X - \_c.X, \_d.Y - \_c.Y);

return Math.Abs(Vector.AngleBetween(cb, cd));

}

}

// Calculating angle D (in degrees)

public double AngleD

{

get

{

Vector da = new(\_a.X - \_d.X, \_a.Y - \_d.Y);

Vector dc = new(\_c.X - \_d.X, \_c.Y - \_d.Y);

return Math.Abs(Vector.AngleBetween(da, dc));

}

}

// AC diagonal as 2 points

public (Point, Point) AC => (A, C);

// BD diagonal as 2 points

public (Point, Point) BD => (B, D);

[JsonProperty("Type")] public string ShapeType => this.GetType().Name;

/// <summary>

/// Default quadrangle constructor.

/// Initializes points with values: A = (0, 0); B = (0, 1); C = (1, 1); D = (1, 0)

/// </summary>

public Quadrangle()

{

\_a = new Point(0, 0);

\_b = new Point(0, 1);

\_c = new Point(1, 1);

\_d = new Point(1, 0);

}

/// <summary>

/// Quadrangle constructor with parameters.

/// </summary>

/// <exception cref="ArgumentException"></exception>

public Quadrangle(Point a, Point b, Point c, Point d)

{

if (!IsQuadrangle(a, b, c, d))

{

throw new ArgumentException("Such quadrangle does not exist.");

}

\_a = new Point(a.X, a.Y);

\_b = new Point(b.X, b.Y);

\_c = new Point(c.X, c.Y);

\_d = new Point(d.X, d.Y);

}

/// <summary>

/// Quadrangle copy constructor

/// </summary>

public Quadrangle(Quadrangle quadrangle) : this(quadrangle.A, quadrangle.B, quadrangle.C, quadrangle.D) { }

public virtual double GetArea()

{

double ac = Math.Sqrt(Math.Pow(AC.Item2.X - AC.Item1.X, 2) + Math.Pow(AC.Item2.Y - AC.Item1.Y, 2));

double bd = Math.Sqrt(Math.Pow(BD.Item2.X - BD.Item1.X, 2) + Math.Pow(BD.Item2.Y - BD.Item1.Y, 2));

Vector acVector = new(\_c.X - \_a.X, \_c.Y - \_a.Y);

Vector bdVector = new(\_d.X - \_b.X, \_d.Y - \_b.Y);

double alpha = Math.Abs(Vector.AngleBetween(acVector, bdVector));

alpha = alpha > 90.0 ? 180.0 - alpha : alpha;

double alphaInRadians = alpha \* Math.PI / 180.0;

double area = ac \* bd \* Math.Sin(alphaInRadians) / 2;

return area;

}

public virtual double GetPerimeter() => AB + AD + BC + CD;

public override string ToString()

{

StringBuilder sb = new();

\_ = sb.Append($"A: ({A}); B: ({B}); C: ({C}) D:({D})|");

\_ = sb.Append($"∠A={AngleA:F2}°; ∠B={AngleB:F2}° ∠C={AngleC:F2}° ∠D={AngleD:F2}°|");

\_ = sb.Append($"AB={AB:F2}; BC={BC:F2}; AD={AD:F2}; CD={CD:F2}|");

\_ = sb.Append($"S={GetArea():F2}; P={GetPerimeter():F2}");

return sb.ToString();

}

public static bool IsQuadrangle(Point a, Point b, Point c, Point d)

{

Vector ab = new(b.X - a.X, b.Y - a.Y);

Vector ad = new(d.X - a.X, d.Y - a.Y);

double angleA = Math.Abs(Vector.AngleBetween(ab, ad));

ab.Negate(); // got ba vector

Vector bc = new(c.X - b.X, c.Y - b.Y);

double angleB = Math.Abs(Vector.AngleBetween(ab, bc));

bc.Negate(); // got cb vector

Vector cd = new(d.X - c.X, d.Y - c.Y);

double angleC = Math.Abs(Vector.AngleBetween(bc, cd));

cd.Negate(); // got dc vector

ad.Negate(); // got da vector

double angleD = Math.Abs(Vector.AngleBetween(cd, ad));

double angleSum = angleA + angleB + angleC + angleD;

if (angleA >= 180.0 || angleB >= 180.0 || angleC >= 180.0 || angleD >= 180.0)

{

return false;

}

if (Math.Abs(360.0 - angleSum) >= 1e-8)

{

return false;

}

return true;

}

public static Quadrangle Parse(string s)

{

if (!Regex.IsMatch(s, @"^\(\d+\.?\d\*,\d+\.?\d\*\) \(\d+\.?\d\*,\d+\.?\d\*\) \(\d+\.?\d\*,\d+\.?\d\*\) \(\d+\.?\d\*,\d+\.?\d\*\)"))

{

throw new FormatException("String does not suit the format.");

}

MatchCollection mPoints = Regex.Matches(s, @"\d+\.?\d\*,\d+\.?\d\*");

List<Point> points = new();

foreach (Match point in mPoints)

{

points.Add(Point.Parse(point.Value));

}

return new Quadrangle(points[0], points[1], points[2], points[3]);

}

public static bool TryParse(string s, out Quadrangle quadrangle)

{

try

{

quadrangle = Quadrangle.Parse(s);

}

catch (Exception)

{

quadrangle = new();

return false;

}

return true;

}

}

}

## 4.7 Rectangle.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text.RegularExpressions;

using System.Windows;

namespace CourseOOP.Models

{

public class Rectangle : Quadrangle

{

public override double GetArea() => AB \* BC;

public override double GetPerimeter() => (AB + BC) \* 2;

/// <summary>

/// Default rectangle constructor

/// </summary>

public Rectangle() : base() { }

/// <summary>

/// Rectangle constructor with parameters.

/// </summary>

/// <exception cref="ArgumentException"></exception>

public Rectangle(Point a, Point b, Point c, Point d) : base(a, b, c, d)

{

if (!IsRectangle(a, b, c, d))

{

throw new ArgumentException("This is not a rectangle.");

}

}

/// <summary>

/// Rectangle copy constructor.

/// </summary>

/// <exception cref="ArgumentException"></exception>

public Rectangle(Rectangle rectangle) : base(rectangle) { }

public static bool IsRectangle(Point a, Point b, Point c, Point d)

{

if (!IsQuadrangle(a, b, c, d))

{

return false;

}

Quadrangle quadrangle = new(a, b, c, d);

List<double> angles = new()

{

quadrangle.AngleA,

quadrangle.AngleB,

quadrangle.AngleC,

quadrangle.AngleD

};

if (angles.Count(angle => Math.Abs(90.0 - angle) <= 1e-8) != angles.Count)

{

return false;

}

return true;

}

public new static Rectangle Parse(string s)

{

if (!Regex.IsMatch(s, @"^\(\d+\.?\d\*,\d+\.?\d\*\) \(\d+\.?\d\*,\d+\.?\d\*\) \(\d+\.?\d\*,\d+\.?\d\*\) \(\d+\.?\d\*,\d+\.?\d\*\)"))

{

throw new FormatException("String does not suit the format.");

}

MatchCollection mPoints = Regex.Matches(s, @"\d+\.?\d\*,\d+\.?\d\*");

List<Point> points = new();

foreach (Match point in mPoints)

{

points.Add(Point.Parse(point.Value));

}

return new Rectangle(points[0], points[1], points[2], points[3]);

}

public static bool TryParse(string s, out Rectangle rectangle)

{

try

{

rectangle = Rectangle.Parse(s);

}

catch (Exception)

{

rectangle = new();

return false;

}

return true;

}

}

}

## 4.8 Trapezium.cs

using System;

using System.Collections.Generic;

using System.Text.RegularExpressions;

using System.Windows;

namespace CourseOOP.Models

{

public class Trapezium : Quadrangle

{

public (double, double) Basis

{

get

{

Vector ab = new(B.X - A.X, B.Y - A.Y);

Vector cd = new(D.X - C.X, D.Y - C.Y);

if (Vector.AngleBetween(ab, cd) <= 1e-8)

{

return (AB, CD);

}

return (BC, CD);

}

}

/// <summary>

/// Default Trapezium constructor

/// </summary>

public Trapezium() : base() { }

/// <summary>

/// Trapezium constructor with parameters.

/// </summary>

/// <exception cref="ArgumentException"></exception>

public Trapezium(Point a, Point b, Point c, Point d) : base(a, b, c, d)

{

if (!IsTrapezium(a, b, c, d))

{

throw new ArgumentException("This is not a trapezium.");

}

}

/// <summary>

/// Trapezium copy constructor.

/// </summary>

/// <exception cref="ArgumentException"></exception>

public Trapezium(Trapezium trapezium) : base (trapezium) { }

public override double GetArea()

{

return base.GetArea();

}

public static bool IsTrapezium(Point a, Point b, Point c, Point d)

{

if (!IsQuadrangle(a, b, c, d))

{

return false;

}

Quadrangle quadrangle = new(a, b, c, d);

Vector ab = new(quadrangle.B.X - quadrangle.A.X, quadrangle.B.Y - quadrangle.A.Y);

Vector bc = new(quadrangle.C.X - quadrangle.B.X, quadrangle.C.Y - quadrangle.B.Y);

Vector cd = new(quadrangle.D.X - quadrangle.C.X, quadrangle.D.Y - quadrangle.C.Y);

Vector ad = new(quadrangle.D.X - quadrangle.A.X, quadrangle.D.Y - quadrangle.A.Y);

return (Vector.AngleBetween(ab, cd) <= 1e-8 &&

!(Vector.AngleBetween(bc, ad) <= 1e-8)) ||

(!(Vector.AngleBetween(ab, cd) <= 1e-8) &&

Vector.AngleBetween(bc, ad) <= 1e-8);

}

public new static Trapezium Parse(string s)

{

if (!Regex.IsMatch(s, @"^\(\d+\.?\d\*,\d+\.?\d\*\) \(\d+\.?\d\*,\d+\.?\d\*\) \(\d+\.?\d\*,\d+\.?\d\*\) \(\d+\.?\d\*,\d+\.?\d\*\)"))

{

throw new FormatException("String does not suit the format.");

}

MatchCollection mPoints = Regex.Matches(s, @"\d+\.?\d\*,\d+\.?\d\*");

List<Point> points = new();

foreach (Match point in mPoints)

{

points.Add(Point.Parse(point.Value));

}

return new Trapezium(points[0], points[1], points[2], points[3]);

}

public static bool TryParse(string s, out Trapezium trapezium)

{

try

{

trapezium = Trapezium.Parse(s);

}

catch (Exception)

{

trapezium = new();

return false;

}

return true;

}

}

}

## 4.9 Hexagon.cs

using System;

using System.Text;

using System.Text.RegularExpressions;

using Newtonsoft.Json;

namespace CourseOOP.Models

{

public class Hexagon : IShape

{

private double \_sideLength;

public double SideLength

{

get => \_sideLength;

set

{

if (value <= 0.0)

{

throw new ArgumentException("Side length must be positive value.", nameof(value));

}

\_sideLength = value;

}

}

[JsonProperty("Type")] public string ShapeType => this.GetType().Name;

/// <summary>

/// Default Hexagon constructor.

/// Initializes SideLength with 1.0

/// </summary>

public Hexagon()

{

\_sideLength = 1.0;

}

/// <summary>

/// Hexagon constructor with parameter.

/// </summary>

/// <exception cref="ArgumentException"></exception>

public Hexagon(double sideLength)

{

SideLength = sideLength;

}

/// <summary>

/// Hexagon copy constructor.

/// </summary>

public Hexagon(Hexagon hexagon)

{

this.SideLength = hexagon.SideLength;

}

public double GetArea() => 3 \* Math.Sqrt(3) \* \_sideLength \* \_sideLength / 2;

public double GetPerimeter() => \_sideLength \* 6;

public override string ToString()

{

StringBuilder sb = new();

\_ = sb.Append($"Side length: {SideLength:F2}| S={GetArea():F2}; P={GetPerimeter():F2}");

return sb.ToString();

}

public static Hexagon Parse(string s)

{

if (!Regex.IsMatch(s, @"^\d+\.?\d\*"))

{

throw new FormatException("String does not suit the format.");

}

return new Hexagon(Double.Parse(s));

}

public static bool TryParse(string s, out Hexagon hexagon)

{

try

{

hexagon = Hexagon.Parse(s);

}

catch (FormatException)

{

hexagon = new Hexagon();

return false;

}

return true;

}

}

}

## 4.10 ShapeHandler.cs

using System;

using System.Collections.Generic;

using System.Diagnostics.CodeAnalysis;

using System.IO;

using System.Linq;

using System.Text;

using CourseOOP.Exceptions;

using Newtonsoft.Json;

using Newtonsoft.Json.Linq;

namespace CourseOOP.Models

{

public class ShapeHandler

{

public List<IShape> Shapes { get; }

[JsonIgnore]

public Dictionary<IShape, StringBuilder> ShapesHistory { get; }

public IShape this[int index]

{

get

{

if (index < 0 || index >= Shapes.Count)

{

throw new IndexOutOfRangeException("Index is out of range.");

}

return Shapes[index];

}

}

/// <summary>

/// Default ShapeHandler constructor

/// </summary>

public ShapeHandler()

{

Shapes = new List<IShape>();

ShapesHistory = new Dictionary<IShape, StringBuilder>();

}

/// <summary>

/// Shape handler constructor with parameter.

/// </summary>

/// <param name="filePath">Path to file with shapes list.</param>

public ShapeHandler(string filePath)

{

Shapes = new List<IShape>();

ShapesHistory = new Dictionary<IShape, StringBuilder>();

ReadFromFile(filePath);

}

/// <summary>

/// ShapeHandler copy constructor.

/// </summary>

public ShapeHandler(ShapeHandler handler)

{

Shapes = handler.Shapes.ToList();

ShapesHistory = handler.ShapesHistory.ToDictionary(entry => entry.Key, entry => entry.Value);

}

/// <summary>

/// Reads shapes list from file.

/// </summary>

/// <param name="filePath">Path to file with shapes list.</param>

/// <exception cref="ArgumentException"></exception>

/// <exception cref="FileNotFoundException"></exception>

/// <exception cref="NotSupportedException"></exception>

/// <exception cref="TypeNotSupportedException"></exception>

public void ReadFromFile(string filePath)

{

if (String.IsNullOrWhiteSpace(filePath))

{

throw new ArgumentException("String is null or whitespace.", nameof(filePath));

}

if (!File.Exists(filePath))

{

throw new FileNotFoundException("File not found.", filePath);

}

FileInfo fileInfo = new(filePath);

if (String.CompareOrdinal(fileInfo.Extension, ".txt") != 0 &&

String.CompareOrdinal(fileInfo.Extension, ".json") != 0)

{

throw new NotSupportedException("Only \*.txt and \*.json files are supported.");

}

string file;

using (StreamReader reader = new(filePath))

{

file = reader.ReadToEnd();

}

if (String.CompareOrdinal(fileInfo.Extension, ".json") == 0)

{

ReadJson(file);

return;

}

using (StreamReader reader = new(filePath))

{

string line;

while ((line = reader.ReadLine()) != null)

{

if (String.IsNullOrWhiteSpace(line))

{

continue;

}

line = line.Trim();

string shapeType = line.Substring(0, line.IndexOf(' '));

IShape shape;

switch (shapeType)

{

case "Triangle":

shape = Triangle.Parse(line.Remove(0, shapeType.Length + 1));

break;

case "RightTriangle":

shape = RightTriangle.Parse(line.Remove(0, shapeType.Length + 1));

break;

case "IsoscelesTriangle":

shape = IsoscelesTriangle.Parse(line.Remove(0, shapeType.Length + 1));

break;

case "EquilateralTriangle":

shape = EquilateralTriangle.Parse(line.Remove(0, shapeType.Length + 1));

break;

case "Quadrangle":

shape = Quadrangle.Parse(line.Remove(0, shapeType.Length + 1));

break;

case "Rectangle":

shape = Rectangle.Parse(line.Remove(0, shapeType.Length + 1));

break;

case "Trapezium":

shape = Rectangle.Parse(line.Remove(0, shapeType.Length + 1));

break;

case "Hexagon":

shape = Hexagon.Parse(line.Remove(0, shapeType.Length + 1));

break;

default:

throw new TypeNotSupportedException("This type of shapes is not supported.");

}

Shapes.Add(shape);

if (!ShapesHistory.ContainsKey(shape))

{

ShapesHistory.Add(shape, new StringBuilder($"{DateTime.Now} - shape was created.\n"));

}

else

{

ShapesHistory[shape].AppendLine($"{DateTime.Now} - shape was created.");

}

}

}

}

/// <summary>

/// Removes shapes at selected index.

/// </summary>

/// <exception cref="IndexOutOfRangeException"></exception>

/// <param name="index"></param>

public void RemoveAt(int index)

{

if (index < 0 || index >= Shapes.Count)

{

throw new IndexOutOfRangeException("Index is out of range.");

}

IShape shape = Shapes.ElementAt(index);

Shapes.RemoveAt(index);

ShapesHistory[shape].AppendLine($"{DateTime.Now} - shape was removed.");

}

/// <summary>

/// Reads shapes list from JSON format.

/// </summary>

/// <exception cref="FormatException"></exception>

/// <exception cref="TypeNotSupportedException"></exception>

/// <param name="json"></param>

public void ReadJson(string json)

{

JObject jObject = JObject.Parse(json);

JToken shapesJToken = jObject["Shapes"] ?? throw new FormatException("Json file must contain attribute \"Shapes\"");

foreach (JToken shapeJToken in shapesJToken)

{

JToken typeJT = shapeJToken["Type"] ?? throw new FormatException("Each token must contain attribute \"Type\".");

string type = typeJT.Value<string>();

IShape shape;

switch (type)

{

case "Triangle":

shape = shapeJToken.ToObject<Triangle>();

Triangle triangle = (Triangle)shape;

if (!Triangle.IsTriangle(triangle.A, triangle.B, triangle.C))

{

throw new InvalidShapeException();

}

break;

case "IsoscelesTriangle":

shape = shapeJToken.ToObject<IsoscelesTriangle>();

IsoscelesTriangle isosceles = (IsoscelesTriangle) shape;

if (!IsoscelesTriangle.IsIsoscelesTriangle(

isosceles.A,

isosceles.B,

isosceles.C

))

{

throw new InvalidShapeException();

}

break;

case "EquilateralTriangle":

shape = shapeJToken.ToObject<EquilateralTriangle>();

EquilateralTriangle equilateral = (EquilateralTriangle) shape;

if (!EquilateralTriangle.IsEquilateralTriangle(

equilateral.A,

equilateral.B,

equilateral.C

))

{

throw new InvalidShapeException();

}

break;

case "RightTriangle":

shape = shapeJToken.ToObject<RightTriangle>();

RightTriangle right = (RightTriangle) shape;

if (!RightTriangle.IsRightTriangle(right.A, right.B, right.C))

{

throw new InvalidShapeException();

}

break;

case "Quadrangle":

shape = shapeJToken.ToObject<Quadrangle>();

Quadrangle quadrangle = (Quadrangle) shape;

if (!Quadrangle.IsQuadrangle(

quadrangle.A,

quadrangle.B,

quadrangle.C,

quadrangle.D

))

{

throw new InvalidShapeException();

}

break;

case "Rectangle":

shape = shapeJToken.ToObject<Rectangle>();

Rectangle rectangle = (Rectangle) shape;

if (!Rectangle.IsRectangle(

rectangle.A,

rectangle.B,

rectangle.C,

rectangle.D

))

{

throw new InvalidShapeException();

}

break;

case "Trapezium":

shape = shapeJToken.ToObject<Trapezium>();

Trapezium trapezium = (Trapezium) shape;

if (!Trapezium.IsTrapezium(

trapezium.A,

trapezium.B,

trapezium.C,

trapezium.D

))

{

throw new InvalidShapeException();

}

break;

case "Hexagon":

shape = shapeJToken.ToObject<Hexagon>();

break;

default:

throw new TypeNotSupportedException("This type of shapes is not supported.");

}

Shapes.Add(shape ?? throw new ShapeParseException("Failed to parse shape from JSON."));

if (!ShapesHistory.ContainsKey(shape))

{

ShapesHistory.Add(shape, new StringBuilder($"{DateTime.Now} - shape was created.\n"));

}

else

{

ShapesHistory[shape].AppendLine($"{DateTime.Now} - shape was created.");

}

}

}

/// <summary>

/// Adds shape to shapes list.

/// </summary>

/// <exception cref="ArgumentNullException"></exception>

/// <param name="shape"></param>

public void AddShape([DisallowNull] IShape shape)

{

if (shape == null)

{

throw new ArgumentNullException(nameof(shape), "Parameter is null.");

}

Shapes.Add(shape);

if (!ShapesHistory.ContainsKey(shape))

{

ShapesHistory.Add(shape, new StringBuilder($"{DateTime.Now} - shape was created.\n"));

}

else

{

\_ = ShapesHistory[shape].AppendLine($"{DateTime.Now} - shape was created.");

}

}

/// <summary>

/// Updates shapes at selected index with new value.

/// </summary>

/// <exception cref="ArgumentNullException"></exception>

/// <param name="shape"></param>

/// <param name="index"></param>

public void UpdateShapeAt([DisallowNull] IShape shape, int index)

{

if (shape == null)

{

throw new ArgumentNullException(nameof(shape), "Argument is null.");

}

if (index < 0 || index >= Shapes.Count)

{

throw new IndexOutOfRangeException("Index is out of range.");

}

StringBuilder data = ShapesHistory[shape];

ShapesHistory.Remove(shape);

Shapes[index] = shape;

ShapesHistory.Add(shape, data);

\_ = data.AppendLine($"{DateTime.Now} - shape was edited.");

}

/// <summary>

/// Writes shapes list to file

/// </summary>

/// <exception cref="ArgumentException"></exception>

/// <exception cref="NotSupportedException"></exception>

/// <param name="filePath">Path to file.</param>

public void WriteToFile(string filePath)

{

if (String.IsNullOrWhiteSpace(filePath))

{

throw new ArgumentException("String is null or whitespace.", nameof(filePath));

}

FileInfo fileInfo = new(filePath);

if (String.CompareOrdinal(fileInfo.Extension, ".txt") != 0 &&

String.CompareOrdinal(fileInfo.Extension, ".json") != 0)

{

throw new NotSupportedException("Only \*.txt and \*.json files are supported.");

}

if (String.CompareOrdinal(fileInfo.Extension, ".json") == 0)

{

using (StreamWriter writer = new(filePath))

{

writer.Write(JsonConvert.SerializeObject(this, Formatting.Indented));

}

}

else

{

using (StreamWriter writer = new(filePath))

{

foreach (IShape shape in this)

{

writer.WriteLine(shape.ToString());

}

}

}

}

/// <summary>

/// Writes shapes history to file.

/// </summary>

/// <exception cref="ArgumentException"></exception>

/// <exception cref="NotSupportedException"></exception>

/// <param name="filePath">Path to file.</param>

public void WriteShapesHistoryToFile(string filePath)

{

if (String.IsNullOrWhiteSpace(filePath))

{

throw new ArgumentException("String is null or whitespace.", nameof(filePath));

}

FileInfo fileInfo = new(filePath);

if (String.CompareOrdinal(fileInfo.Extension, ".txt") != 0)

{

throw new NotSupportedException("Only \*.txt files are supported.");

}

using StreamWriter writer = new(filePath);

foreach ((IShape shape, StringBuilder history) in ShapesHistory)

{

writer.WriteLine($"Shape: {shape.ShapeType} Information: {shape}\n");

writer.WriteLine($"History: {history}\n");

}

}

public IEnumerator<IShape> GetEnumerator() => Shapes.GetEnumerator();

}

}

## 4.11 MainWindow.xaml.cs

using System;

using System.Linq;

using System.Windows;

using System.Windows.Input;

using CourseOOP.Exceptions;

using CourseOOP.Models;

using Microsoft.Win32;

namespace CourseOOP.Views

{

public class DataGridRecord

{

public string ShapeType { get; set; }

public string Information { get; set; }

}

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

public ShapeHandler ShapeHandler { get; }

public MainWindow()

{

InitializeComponent();

ShapeHandler = new();

}

private void ListViewItem\_MouseEnter(object sender, MouseEventArgs e)

{

// Set tooltip visibility

if (TgBtn.IsChecked == true)

{

ttRead.Visibility = Visibility.Collapsed;

ttWrite.Visibility = Visibility.Collapsed;

ttAdd.Visibility = Visibility.Collapsed;

ttEdit.Visibility = Visibility.Collapsed;

ttRemove.Visibility = Visibility.Collapsed;

ttShow.Visibility = Visibility.Collapsed;

}

else

{

ttRead.Visibility = Visibility.Visible;

ttWrite.Visibility = Visibility.Visible;

ttAdd.Visibility = Visibility.Visible;

ttEdit.Visibility = Visibility.Visible;

ttRemove.Visibility = Visibility.Visible;

ttShow.Visibility = Visibility.Visible;

}

}

private void TgBtn\_Unchecked(object sender, RoutedEventArgs e)

{

GradientBG.Opacity = 1;

ShapesGrid.Opacity = 1;

ShapesGrid.IsEnabled = true;

}

private void TgBtn\_Checked(object sender, RoutedEventArgs e)

{

GradientBG.Opacity = 0.3;

ShapesGrid.Opacity = 0.3;

ShapesGrid.IsEnabled = false;

}

private void BG\_PreviewMouseLeftButtonDown(object sender, MouseButtonEventArgs e)

{

TgBtn.IsChecked = false;

}

private void CloseBtn\_Click(object sender, RoutedEventArgs e)

{

Close();

}

/// <summary>

/// Updating grid after changing shape list.

/// </summary>

public void UpdateGrid()

{

ShapesGrid.Items.Clear();

foreach (IShape shape in ShapeHandler)

{

string shapeType = shape.ShapeType;

string information = shape.ToString();

DataGridRecord record = new()

{

ShapeType = shapeType,

Information = information

};

ShapesGrid.Items.Add(record);

}

}

/// <summary>

/// Reading from file: either JSON or TXT

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void StackPanel\_MouseDown(object sender, MouseButtonEventArgs e)

{

OpenFileDialog dialog = new()

{

Filter = "Text files (\*.txt)|\*.txt|JSON files (\*.json)|\*.json"

};

if (dialog.ShowDialog() == true)

{

try

{

ShapeHandler.ReadFromFile(dialog.FileName);

}

catch (InvalidShapeException)

{

\_ = MessageBox.Show(

this,

"Failed to read some shape from the file you have chosen. Check your file and try again.",

"Error.",

MessageBoxButton.OK

);

}

catch (TypeNotSupportedException)

{

\_ = MessageBox.Show(

this,

"The file you have chosen contains shape of type that is not currently supported.",

"Error.",

MessageBoxButton.OK

);

}

catch (FormatException)

{

\_ = MessageBox.Show(

this,

"Failed to read some shape from the file you have chosen. Check your file and try again.",

"Error.",

MessageBoxButton.OK

);

}

catch (ArgumentException ex)

{

\_ = MessageBox.Show(this,

$"File contains invalid shape values. Message text: {ex.Message}",

"Error.",

MessageBoxButton.OK

);

}

finally

{

UpdateGrid();

}

}

}

/// <summary>

/// Saving shapes to file: either TXT or JSON.

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void StackPanel\_MouseDown\_1(object sender, MouseButtonEventArgs e)

{

if (!ShapeHandler.Shapes.Any())

{

\_ = MessageBox.Show(this, "Nothing to save. Shapes list is empty.", "Message.", MessageBoxButton.OK);

return;

}

SaveFileDialog dialog = new()

{

Filter = "Text Files (\*.txt)|\*.txt|JSON Files (\*.json)|\*.json"

};

if (dialog.ShowDialog() == true)

{

ShapeHandler.WriteToFile(dialog.FileName);

}

}

/// <summary>

/// Adding new shape to grid.

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void StackPanel\_MouseDown\_2(object sender, MouseButtonEventArgs e)

{

EditingPagesFrame.Content = new AddingPage(this);

}

/// <summary>

/// Editing existing shape.

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void StackPanel\_MouseDown\_3(object sender, MouseButtonEventArgs e)

{

if (!ShapeHandler.Shapes.Any())

{

\_ = MessageBox.Show(this, "Nothing to edit yet.", "Message.", MessageBoxButton.OK);

return;

}

EditingPagesFrame.Content = new EditingPage(this);

}

/// <summary>

/// Removing shape from grid.

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void StackPanel\_MouseDown\_4(object sender, MouseButtonEventArgs e)

{

if (!ShapeHandler.Shapes.Any())

{

\_ = MessageBox.Show(this, "Nothing to remove yet.", "Message.", MessageBoxButton.OK);

return;

}

EditingPagesFrame.Content = new RemovingPage(this);

}

/// <summary>

/// Saving shapes' history.

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void StackPanel\_MouseDown\_5(object sender, MouseButtonEventArgs e)

{

if (!ShapeHandler.Shapes.Any())

{

\_ = MessageBox.Show(this, "Nothing to save. Shapes list is empty.", "Message.", MessageBoxButton.OK);

return;

}

SaveFileDialog dialog = new()

{

Filter = "Text Files (\*.txt)|\*.txt"

};

if (dialog.ShowDialog() == true)

{

ShapeHandler.WriteShapesHistoryToFile(dialog.FileName);

}

}

}

}

## 4.12 AddingPage.xaml.cs

using System;

using System.Text.RegularExpressions;

using System.Windows;

using System.Windows.Controls;

using CourseOOP.Models;

using Rectangle = CourseOOP.Models.Rectangle;

namespace CourseOOP.Views

{

/// <summary>

/// Interaction logic for AddingPage.xaml

/// </summary>

public partial class AddingPage : Page

{

private MainWindow \_parent;

public AddingPage()

{

InitializeComponent();

\_parent = null;

}

public AddingPage(MainWindow parent)

{

InitializeComponent();

\_parent = parent;

}

private void AddBtn\_Click(object sender, RoutedEventArgs e)

{

string regexPatternPoint = @"^(\(\d+(,|;)\s\*\d+\))|^(\d+(,|;)\s\*\d+)";

switch (ShapeTypeBox.SelectedIndex)

{

case <= 3:

if (!Regex.IsMatch(ATextBox.Text, regexPatternPoint))

{

\_ = MessageBox.Show(\_parent, "Point A is invalid.", "Error.", MessageBoxButton.OK);

}

else if (!Regex.IsMatch(BTextBox.Text, regexPatternPoint))

{

\_ = MessageBox.Show(\_parent, "Point B is invalid.", "Error.", MessageBoxButton.OK);

}

else if (!Regex.IsMatch(CTextBox.Text, regexPatternPoint))

{

\_ = MessageBox.Show(\_parent, "Point C is invalid.", "Error.", MessageBoxButton.OK);

}

else

{

Point a = Point.Parse(Regex.Match(ATextBox.Text, @"(\d+(,|;)\s\*\d+)").Value.Replace(";", ","));

Point b = Point.Parse(Regex.Match(BTextBox.Text, @"(\d+(,|;)\s\*\d+)").Value.Replace(";", ","));

Point c = Point.Parse(Regex.Match(CTextBox.Text, @"(\d+(,|;)\s\*\d+)").Value.Replace(";", ","));

if (!Triangle.IsTriangle(a, b, c))

{

\_ = MessageBox.Show(\_parent, "This is not a triangle.", "Error.", MessageBoxButton.OK);

}

else

{

int index = ShapeTypeBox.SelectedIndex;

Triangle triangle;

if (index == 0)

{

triangle = new(a, b, c);

}

else if (index == 1)

{

if (!RightTriangle.IsRightTriangle(a, b, c))

{

\_ = MessageBox.Show(\_parent, "This is not a right triangle.", "Error.", MessageBoxButton.OK);

return;

}

triangle = new RightTriangle(a, b, c);

}

else if (index == 2)

{

if (!IsoscelesTriangle.IsIsoscelesTriangle(a, b, c))

{

\_ = MessageBox.Show(\_parent, "This is not an isosceles triangle.", "Error.", MessageBoxButton.OK);

return;

}

triangle = new IsoscelesTriangle(a, b, c);

}

else

{

if (!EquilateralTriangle.IsEquilateralTriangle(a, b, c))

{

\_ = MessageBox.Show(\_parent, "This is not an equilateral triangle.", "Error.", MessageBoxButton.OK);

return;

}

triangle = new EquilateralTriangle(a, b, c);

}

\_parent.ShapeHandler.AddShape(triangle);

\_parent.UpdateGrid();

\_parent.EditingPagesFrame.Content = null;

}

}

break;

case <= 6:

if (!Regex.IsMatch(ATextBox.Text, regexPatternPoint))

{

\_ = MessageBox.Show(\_parent, "Point A is invalid.", "Error.", MessageBoxButton.OK);

}

else if (!Regex.IsMatch(BTextBox.Text, regexPatternPoint))

{

\_ = MessageBox.Show(\_parent, "Point B is invalid.", "Error.", MessageBoxButton.OK);

}

else if (!Regex.IsMatch(CTextBox.Text, regexPatternPoint))

{

\_ = MessageBox.Show(\_parent, "Point C is invalid.", "Error.", MessageBoxButton.OK);

}

else if (!Regex.IsMatch(DTextBox.Text, regexPatternPoint))

{

\_ = MessageBox.Show(\_parent, "Point D is invalid.", "Error.", MessageBoxButton.OK);

}

else

{

Point a = Point.Parse(Regex.Match(ATextBox.Text, @"(\d+(,|;)\s\*\d+)").Value.Replace(";", ","));

Point b = Point.Parse(Regex.Match(BTextBox.Text, @"(\d+(,|;)\s\*\d+)").Value.Replace(";", ","));

Point c = Point.Parse(Regex.Match(CTextBox.Text, @"(\d+(,|;)\s\*\d+)").Value.Replace(";", ","));

Point d = Point.Parse(Regex.Match(DTextBox.Text, @"(\d+(,|;)\s\*\d+)").Value.Replace(";", ","));

int index = ShapeTypeBox.SelectedIndex;

Quadrangle quadrangle;

if (index == 4)

{

if (!Quadrangle.IsQuadrangle(a, b, c, d))

{

\_ = MessageBox.Show(\_parent, "This is not a quadrangle.", "Error.", MessageBoxButton.OK);

return;

}

quadrangle = new Quadrangle(a, b, c, d);

}

else if (index == 5)

{

if (!Rectangle.IsRectangle(a, b, c, d))

{

\_ = MessageBox.Show(\_parent, "This is not a rectangle.", "Error.", MessageBoxButton.OK);

return;

}

quadrangle = new Rectangle(a, b, c, d);

}

else

{

if (!Trapezium.IsTrapezium(a, b, c, d))

{

\_ = MessageBox.Show(\_parent, "This is not a trapezium.", "Error.", MessageBoxButton.OK);

return;

}

quadrangle = new Trapezium(a, b, c, d);

}

\_parent.ShapeHandler.AddShape(quadrangle);

\_parent.UpdateGrid();

\_parent.EditingPagesFrame.Content = null;

}

break;

case 7:

if (!Double.TryParse(SideTextBox.Text, out double sideLength))

{

\_ = MessageBox.Show(\_parent, "Failed to parse side length.", "Error.");

return;

}

if (sideLength <= 0.0)

{

\_ = MessageBox.Show(\_parent, "Side length cannot be negative value.", "Error.");

return;

}

Hexagon hexagon = new(sideLength);

\_parent.ShapeHandler.AddShape(hexagon);

\_parent.UpdateGrid();

\_parent.EditingPagesFrame.Content = null;

break;

default:

\_ = MessageBox.Show(\_parent, "Select shape type first.", "Message", MessageBoxButton.OK);

break;

}

}

private void CancelBtn\_Click(object sender, RoutedEventArgs e)

{

\_parent.EditingPagesFrame.Content = null;

}

private void ShapeTypeBox\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

int selectedIndex = ShapeTypeBox.SelectedIndex;

switch (selectedIndex)

{

case <= 3:

ATextBox.IsEnabled = true;

BTextBox.IsEnabled = true;

CTextBox.IsEnabled = true;

DTextBox.IsEnabled = false;

SideTextBox.IsEnabled = false;

break;

case <= 6:

ATextBox.IsEnabled = true;

BTextBox.IsEnabled = true;

CTextBox.IsEnabled = true;

DTextBox.IsEnabled = true;

SideTextBox.IsEnabled = false;

break;

case 7:

ATextBox.IsEnabled = false;

BTextBox.IsEnabled = false;

CTextBox.IsEnabled = false;

DTextBox.IsEnabled = false;

SideTextBox.IsEnabled = true;

break;

}

}

}

}

## 4.13 EditingPage.xaml.cs

using System;

using System.Text.RegularExpressions;

using System.Windows;

using System.Windows.Controls;

using CourseOOP.Models;

using Rectangle = CourseOOP.Models.Rectangle;

namespace CourseOOP.Views

{

/// <summary>

/// Interaction logic for EditingPage.xaml

/// </summary>

public partial class EditingPage : Page

{

private MainWindow \_parent;

public EditingPage()

{

InitializeComponent();

}

public EditingPage(MainWindow parent)

{

InitializeComponent();

\_parent = parent;

ShapeTypeBox.ItemsSource = parent.ShapeHandler.Shapes;

}

private void EditBtn\_OnClick(object sender, RoutedEventArgs e)

{

string regexPatternPoint = @"^(\(\d+(,|;)\s\*\d+\))|^(\d+(,|;)\s\*\d+)";

int index = ShapeTypeBox.SelectedIndex;

if (index < 0)

{

\_ = MessageBox.Show(\_parent, "Select shape to edit first.", "Message.", MessageBoxButton.OK);

return;

}

IShape shapeToEdit = \_parent.ShapeHandler.Shapes[index];

if (shapeToEdit is Triangle)

{

if (!Regex.IsMatch(ATextBox.Text, regexPatternPoint))

{

\_ = MessageBox.Show(\_parent, "Point A is invalid.", "Error.", MessageBoxButton.OK);

}

else if (!Regex.IsMatch(BTextBox.Text, regexPatternPoint))

{

\_ = MessageBox.Show(\_parent, "Point B is invalid.", "Error.", MessageBoxButton.OK);

}

else if (!Regex.IsMatch(CTextBox.Text, regexPatternPoint))

{

\_ = MessageBox.Show(\_parent, "Point C is invalid.", "Error.", MessageBoxButton.OK);

}

else

{

Point a = Point.Parse(Regex.Match(ATextBox.Text, @"(\d+(,|;)\s\*\d+)").Value.Replace(";", ","));

Point b = Point.Parse(Regex.Match(BTextBox.Text, @"(\d+(,|;)\s\*\d+)").Value.Replace(";", ","));

Point c = Point.Parse(Regex.Match(CTextBox.Text, @"(\d+(,|;)\s\*\d+)").Value.Replace(";", ","));

if (!Triangle.IsTriangle(a, b, c))

{

\_ = MessageBox.Show(\_parent, "This is not a triangle.", "Error.", MessageBoxButton.OK);

}

else

{

switch (shapeToEdit)

{

case RightTriangle when !RightTriangle.IsRightTriangle(a, b, c):

\_ = MessageBox.Show(\_parent, "This is not a right triangle.", "Error.", MessageBoxButton.OK);

return;

case IsoscelesTriangle when !IsoscelesTriangle.IsIsoscelesTriangle(a, b, c):

\_ = MessageBox.Show(\_parent, "This is not an isosceles triangle.", "Error.", MessageBoxButton.OK);

return;

case EquilateralTriangle when !EquilateralTriangle.IsEquilateralTriangle(a, b, c):

\_ = MessageBox.Show(\_parent, "This is not an equilateral triangle.", "Error.", MessageBoxButton.OK);

return;

}

(shapeToEdit as Triangle).A = new(a.X, a.Y);

(shapeToEdit as Triangle).B = new(b.X, b.Y);

(shapeToEdit as Triangle).C = new(c.X, c.Y);

}

}

}

else if (shapeToEdit is Quadrangle)

{

if (!Regex.IsMatch(ATextBox.Text, regexPatternPoint))

{

\_ = MessageBox.Show(\_parent, "Point A is invalid.", "Error.", MessageBoxButton.OK);

}

else if (!Regex.IsMatch(BTextBox.Text, regexPatternPoint))

{

\_ = MessageBox.Show(\_parent, "Point B is invalid.", "Error.", MessageBoxButton.OK);

}

else if (!Regex.IsMatch(CTextBox.Text, regexPatternPoint))

{

\_ = MessageBox.Show(\_parent, "Point C is invalid.", "Error.", MessageBoxButton.OK);

}

else if (!Regex.IsMatch(DTextBox.Text, regexPatternPoint))

{

\_ = MessageBox.Show(\_parent, "Point D is invalid.", "Error.", MessageBoxButton.OK);

}

else

{

Point a = Point.Parse(Regex.Match(ATextBox.Text, @"(\d+(,|;)\s\*\d+)").Value.Replace(";", ","));

Point b = Point.Parse(Regex.Match(BTextBox.Text, @"(\d+(,|;)\s\*\d+)").Value.Replace(";", ","));

Point c = Point.Parse(Regex.Match(CTextBox.Text, @"(\d+(,|;)\s\*\d+)").Value.Replace(";", ","));

Point d = Point.Parse(Regex.Match(DTextBox.Text, @"(\d+(,|;)\s\*\d+)").Value.Replace(";", ","));

if (!Quadrangle.IsQuadrangle(a, b, c, d))

{

\_ = MessageBox.Show(\_parent, "This is not a quadrangle.", "Error.", MessageBoxButton.OK);

}

else

{

switch (shapeToEdit)

{

case Rectangle when !Rectangle.IsRectangle(a, b, c, d):

\_ = MessageBox.Show(\_parent, "This is not a rectangle.", "Error.", MessageBoxButton.OK);

return;

case Trapezium when !Trapezium.IsTrapezium(a, b, c, d):

\_ = MessageBox.Show(\_parent, "This is not a trapezium.", "Error.", MessageBoxButton.OK);

return;

}

}

(shapeToEdit as Quadrangle).A = new(a.X, a.Y);

(shapeToEdit as Quadrangle).B = new(b.X, b.Y);

(shapeToEdit as Quadrangle).C = new(c.X, c.Y);

(shapeToEdit as Quadrangle).D = new(d.X, d.Y);

}

}

else

{

if (!Double.TryParse(SideTextBox.Text, out double sideLength))

{

\_ = MessageBox.Show(\_parent, "Failed to parse side length.", "Error.");

return;

}

if (sideLength <= 0.0)

{

\_ = MessageBox.Show(\_parent, "Side length cannot be negative value.", "Error.");

return;

}

(shapeToEdit as Hexagon).SideLength = sideLength;

}

\_parent.ShapeHandler.UpdateShapeAt(shapeToEdit, index);

\_parent.UpdateGrid();

\_parent.EditingPagesFrame.Content = null;

}

private void CancelBtn\_Click(object sender, RoutedEventArgs e)

{

\_parent.EditingPagesFrame.Content = null;

}

private void ShapeTypeBox\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

int index = ShapeTypeBox.SelectedIndex;

IShape shapeToEdit = \_parent.ShapeHandler.Shapes[index];

if (shapeToEdit is Triangle)

{

ATextBox.IsEnabled = true;

BTextBox.IsEnabled = true;

CTextBox.IsEnabled = true;

DTextBox.IsEnabled = false;

SideTextBox.IsEnabled = false;

ATextBox.Text = $"({(shapeToEdit as Triangle).A})";

BTextBox.Text = $"({(shapeToEdit as Triangle).B})";

CTextBox.Text = $"({(shapeToEdit as Triangle).C})";

DTextBox.Text = "";

SideTextBox.Text = "";

}

else if (shapeToEdit is Quadrangle)

{

ATextBox.IsEnabled = true;

BTextBox.IsEnabled = true;

CTextBox.IsEnabled = true;

DTextBox.IsEnabled = true;

SideTextBox.IsEnabled = false;

ATextBox.Text = $"({(shapeToEdit as Quadrangle).A})";

BTextBox.Text = $"({(shapeToEdit as Quadrangle).B})";

CTextBox.Text = $"({(shapeToEdit as Quadrangle).C})";

DTextBox.Text = $"({(shapeToEdit as Quadrangle).D})";

SideTextBox.Text = "";

}

else

{

ATextBox.IsEnabled = false;

BTextBox.IsEnabled = false;

CTextBox.IsEnabled = false;

DTextBox.IsEnabled = false;

SideTextBox.IsEnabled = true;

ATextBox.Text = "";

BTextBox.Text = "";

CTextBox.Text = "";

DTextBox.Text = "";

SideTextBox.Text = $"{(shapeToEdit as Hexagon).SideLength}";

}

}

}

}

## 4.14 RemovingPage.xaml.cs

using System.Windows;

using System.Windows.Controls;

namespace CourseOOP.Views

{

/// <summary>

/// Interaction logic for RemovingPage.xaml

/// </summary>

public partial class RemovingPage : Page

{

private MainWindow \_parent;

public RemovingPage()

{

InitializeComponent();

}

public RemovingPage(MainWindow parent)

{

InitializeComponent();

\_parent = parent;

ShapeBox.ItemsSource = \_parent.ShapeHandler.Shapes;

}

private void CancelBtn\_Click(object sender, RoutedEventArgs e)

{

\_parent.EditingPagesFrame.Content = null;

}

private void RemoveBtn\_Click(object sender, RoutedEventArgs e)

{

int index = ShapeBox.SelectedIndex;

if (index < 0)

{

\_ = MessageBox.Show(\_parent, "Select shape to remove first.", "Message.", MessageBoxButton.OK);

return;

}

\_parent.ShapeHandler.RemoveAt(index);

\_parent.UpdateGrid();

\_parent.EditingPagesFrame.Content = null;

}

}

}

# 

# **Протокол роботи**

Програма може зчитувати список книг з файлів з розширенням json або txt.

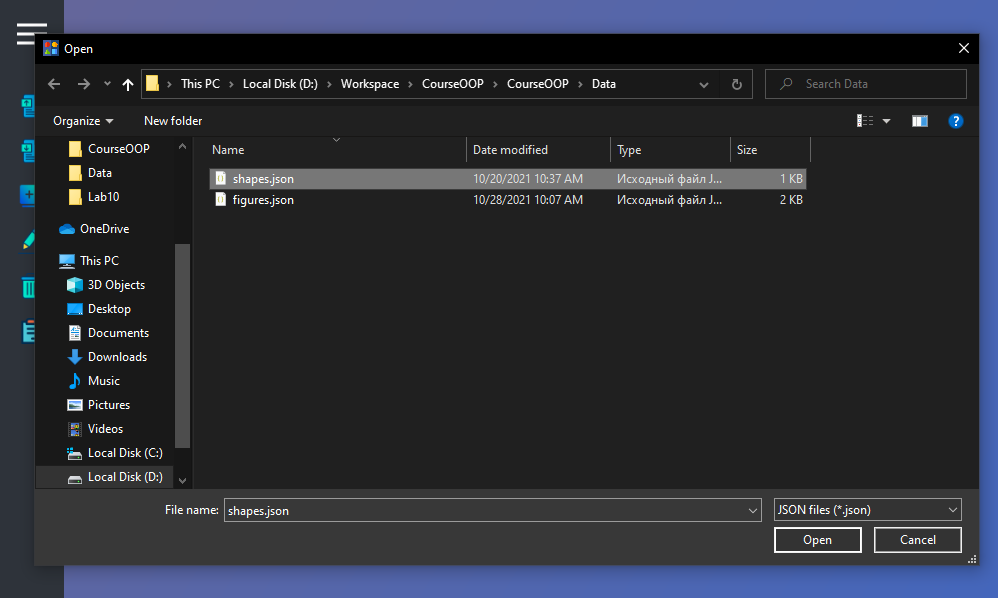


Рис. 5. 1. Вибір файлу із розширенням json.

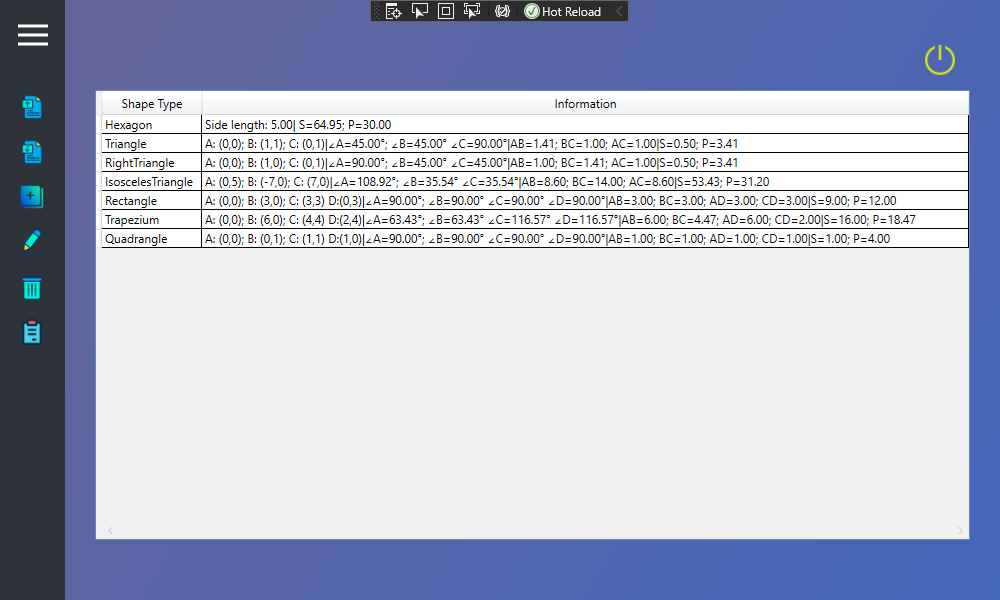


Рис. 5. 2. Фігури з файлу відобразились в таблиці.

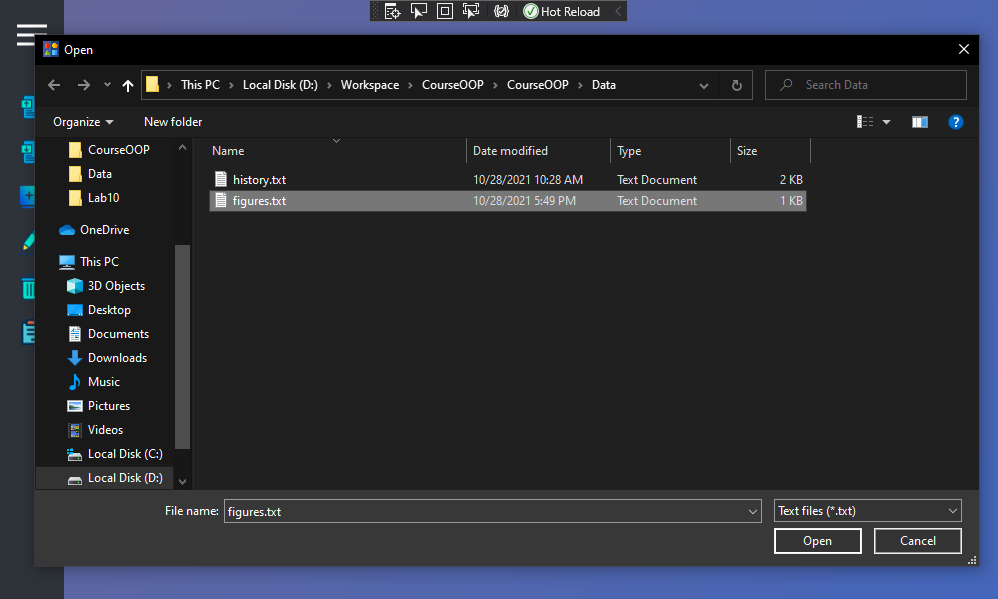


Рис. 5. 3. Вибір файлу із розширенням txt.

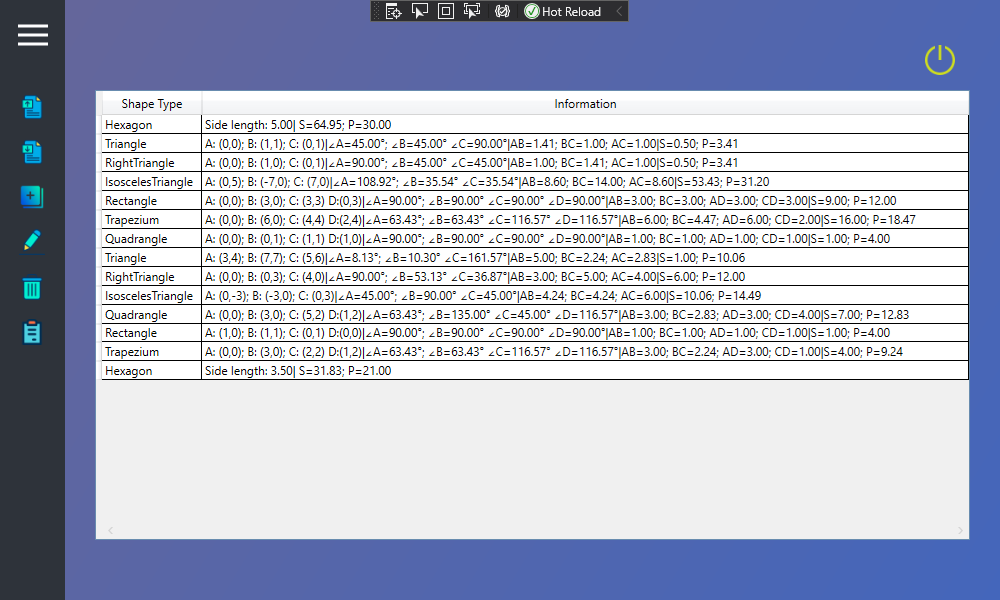


Рис. 5. 4. Фігури з файлу відобразились в таблиці.

Програма автоматично відображає значення периметру і площі в таблиці, як видно на рисунках вище.

Програма дозволяє додати нову фігуру до списку вручну.

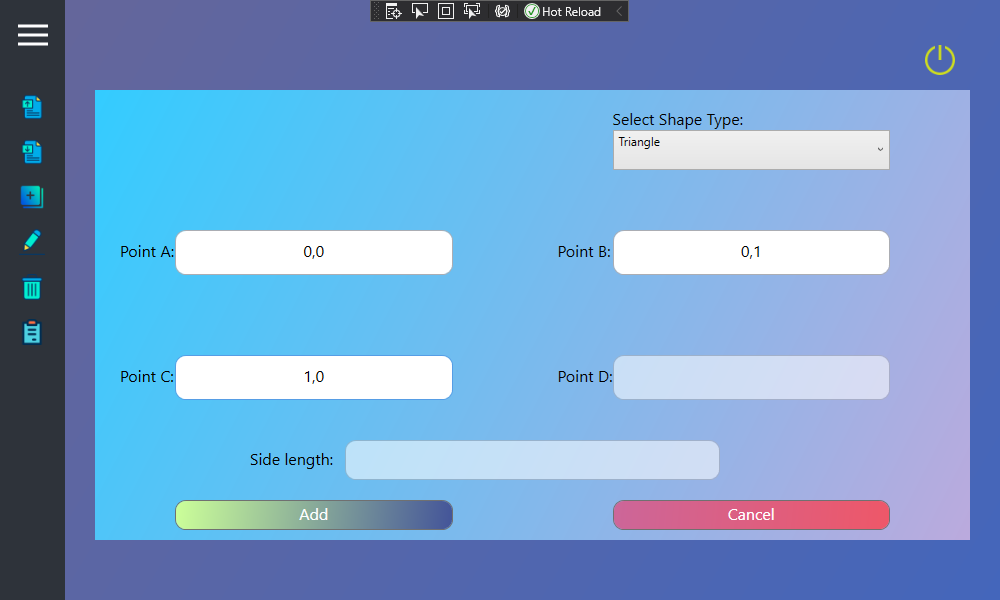


Рис. 5. 5. Форма для додавання фігури.

Програма дозволяє змінити значення полів існуючої фігури.

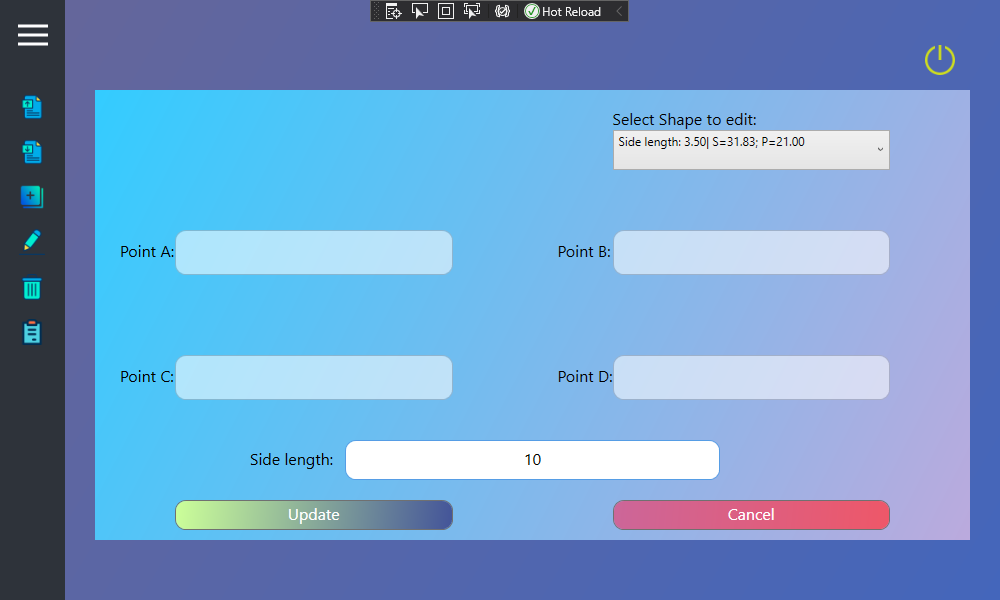


Рис. 5. 6.. Форма для редагування фігури зі списку.

Програма дозволяє видаляти фігуру зі списку.

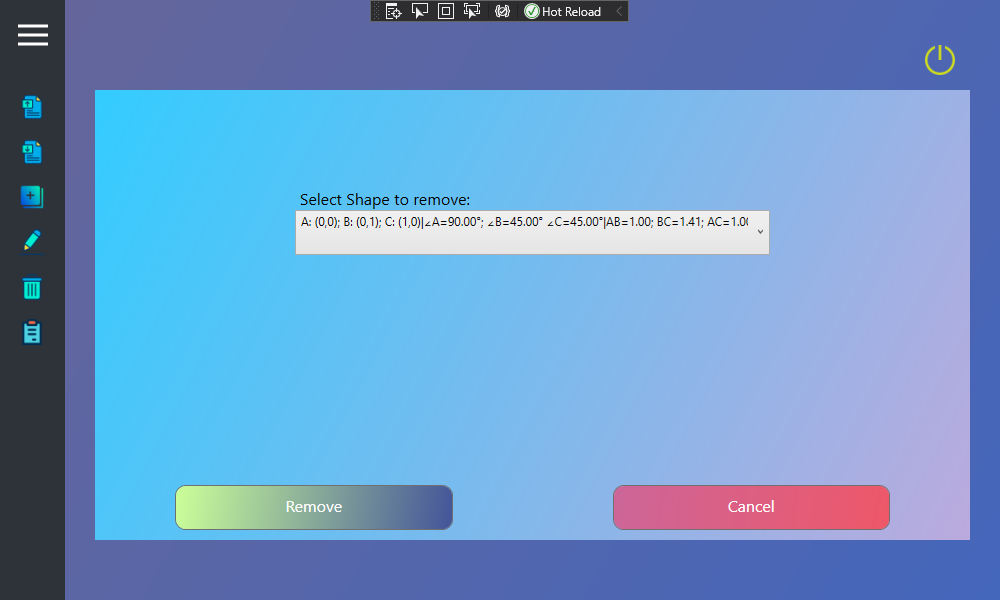


Рис. 5. 7. Вибір фігури зі списку, яку треба видалити.

Програма веде історію по кожній з фігур і дозволяє зберегти її у текстовий файл.

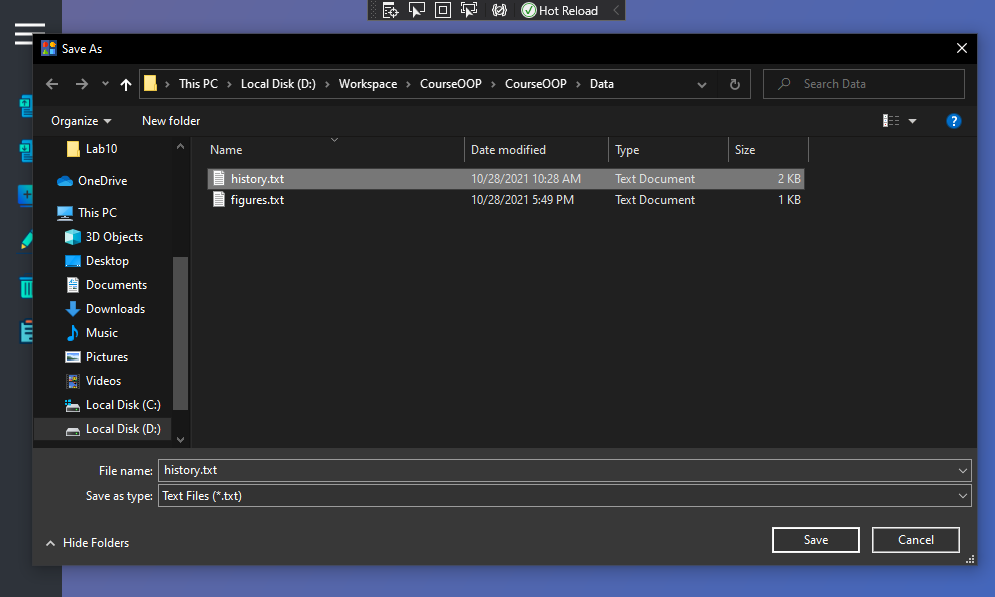


Рис. 5. 8. Вибір файлу, в який потрібно зберегти історію фігур.

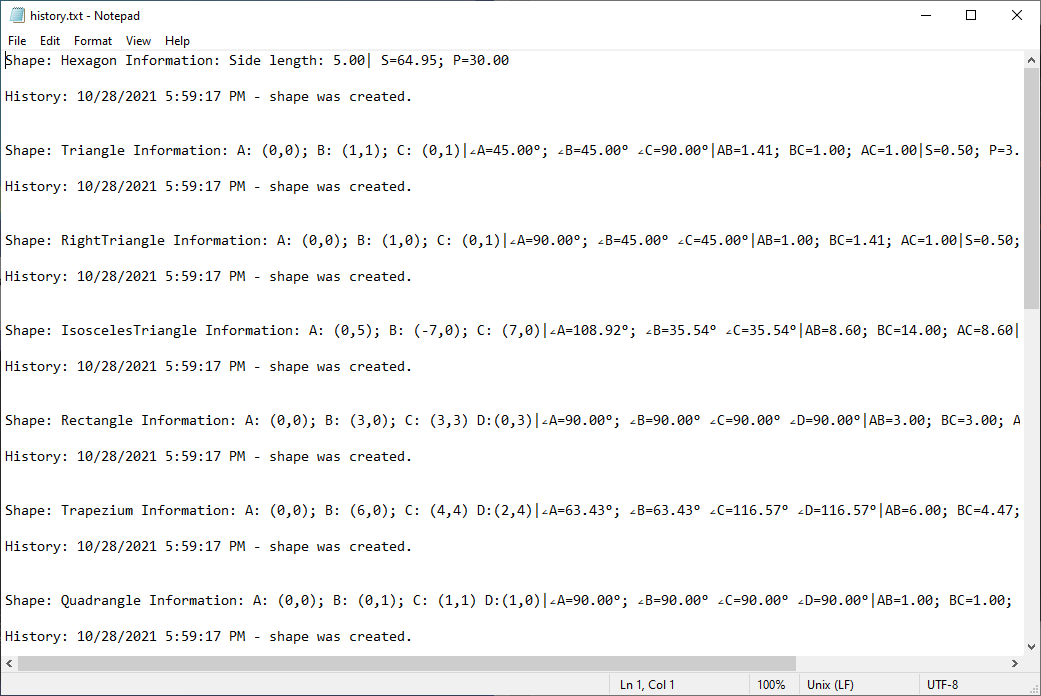


Рис. 5. 9. Файл з історією фігур.

# **Інструкція користувача**

1. **Компоненти ПЗ**

Програму розроблено на мові програмування С# у середовищі розробки Microsoft Visual Studio 2019 із використанням технології WPF і може експлуатуватися під управлінням сімейства операційних систем Windows. Проектування підсистем відбувалось відповідно до об’єктно-орієнтованого підходу. Всі класи документувались інформаційно і семантично.

Для коректної роботи програми необхідна користувацька машина з процесором з тактовою частотою не менше, ніж 500MHz та оперативною пам’яттю не менше, ніж 256 MB. Потрібно 10 MB вільного місця на диску Перелік необхідних файлів для коректного запуску програми навелено у таблиці:

|  |  |  |  |
| --- | --- | --- | --- |
| № | Файл | Призначення | Належить проекту |
| 1. | ShapesApp.exe | Виконавчий файл програми | ShapesApp |
| 2. | ShapesApp.dll | Бібліотека ядра | ShapesApp |
| 3. | Newtonsoft.Json.dll | Бібліотека, яка дозволяє обробляти json-файли | ShapesApp |

1. **Встановлення ПЗ**

Для роботи програми необхідно запустити файл ShapesApp.exe. Нічого

додаткового встановлювати не потрібно.

1. **Базові функції ПЗ**

* Для відкриття файлу із фігурами, потрібно в меню, яке знаходиться зліва натиснути на кнопку «Read from file» (перша кнопка в меню), відкриється файловий провідник, де ви зможете вибрати файл розширення txt або json. Після того, як вибрали, натисніть ОК і фігури з’являться у списку.

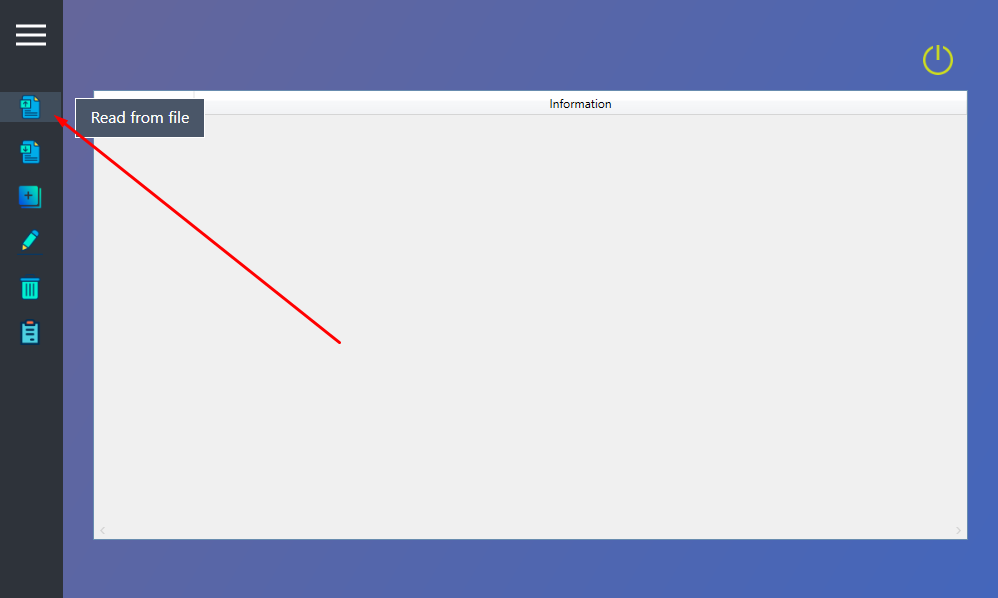


Рис. 6. 1. Кнопка «Read from file»

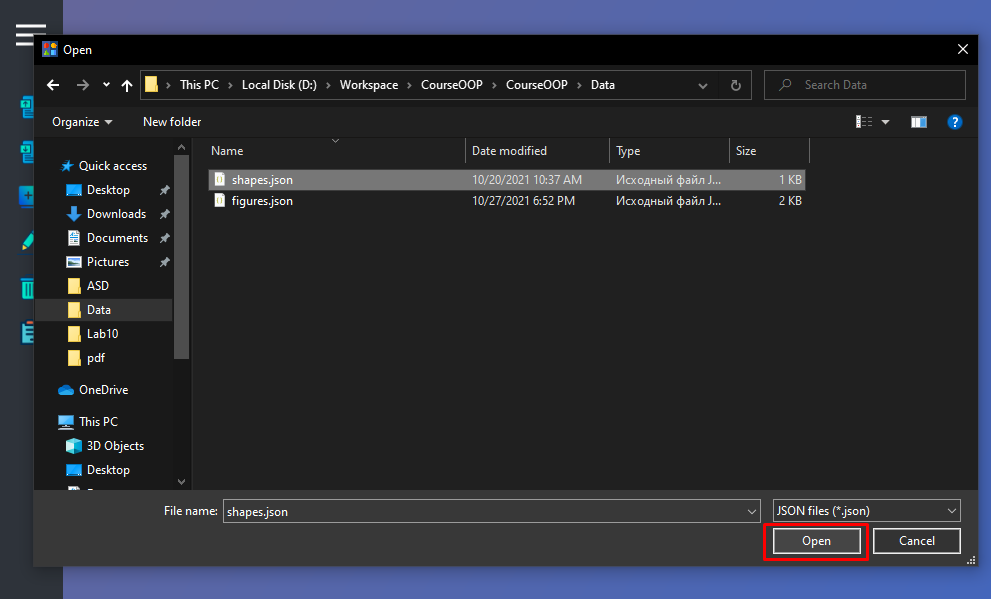


Рис. 6. 2. Вибір файлу.

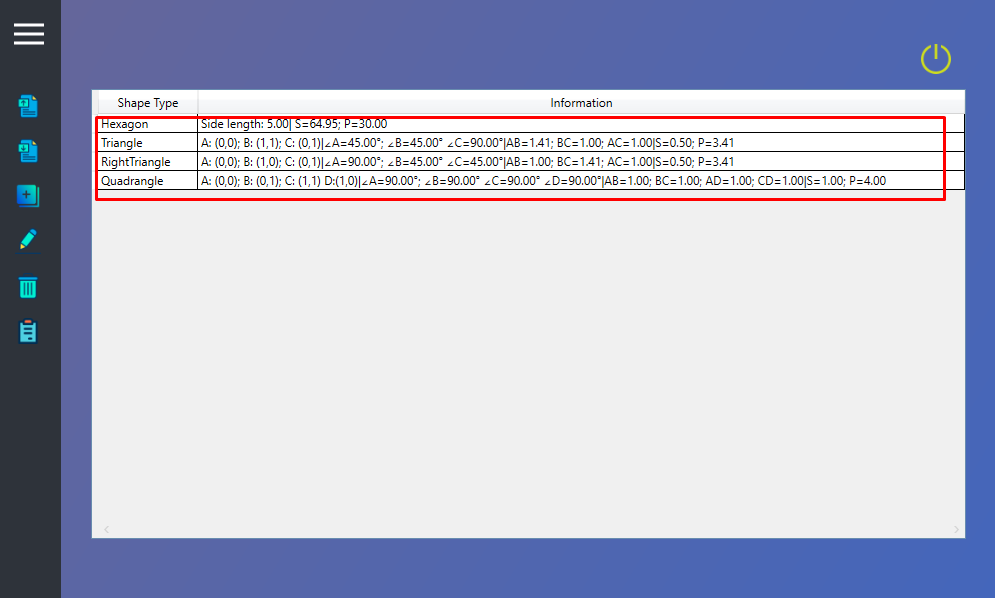


Рис. 6. 3. Фігури з файлу відобразились в таблиці.

* Для збереження фігур у файл, потрібно в меню натиснути на кнопку «Write to file» (друга кнопка в меню), відкриється файловий провідник, де ви зможете вибрати файл розширення txt або json. Після того, як вибрали, натисність ОК і фігури збережуться у файлі.

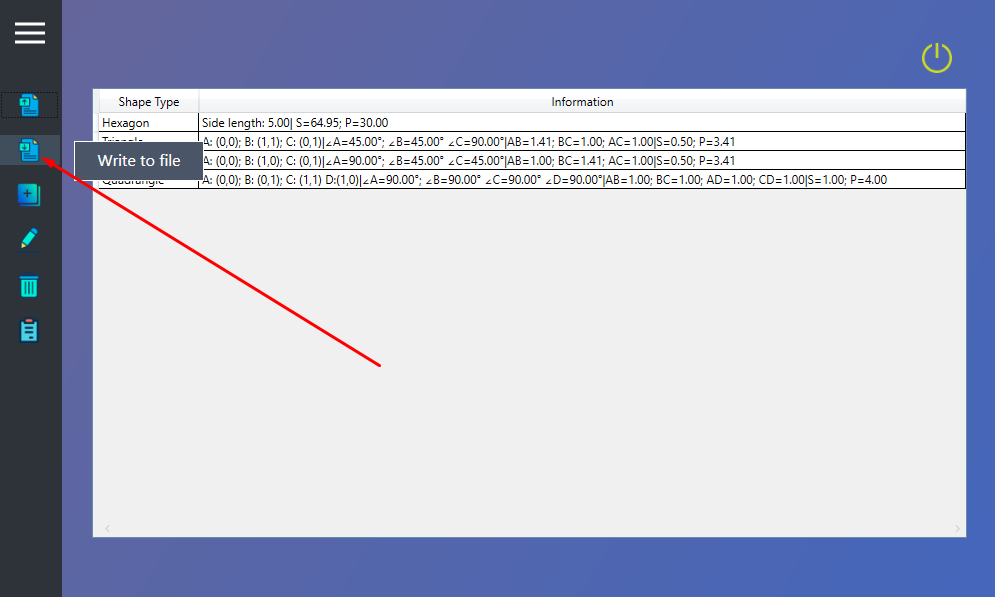


Рис. 6. 4. Кнопка «Write to file»

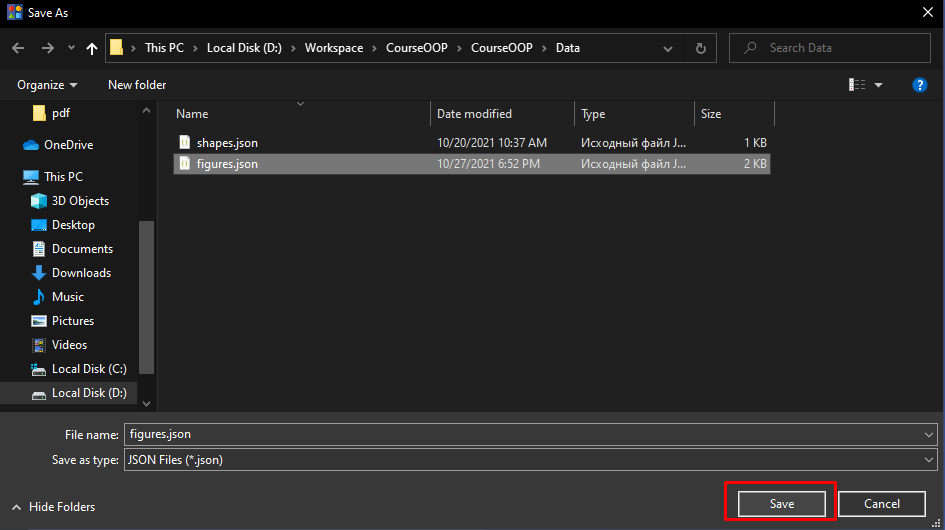


Рис. 6. 5. Вибір файлу для запису.

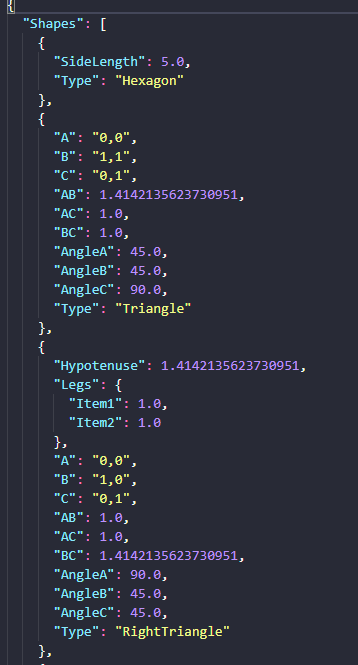


Рис. 6. 6. Результат.

* Щоб додати нову фігуру вручну потрібно в меню натиснути на кнопку «New Shape» (третя кнопка в меню), з’явиться форма де потрібно вибрати тип фігури, яку хочете додати, заповнити всі необхідні поля і натиснути на кнопку «Add».

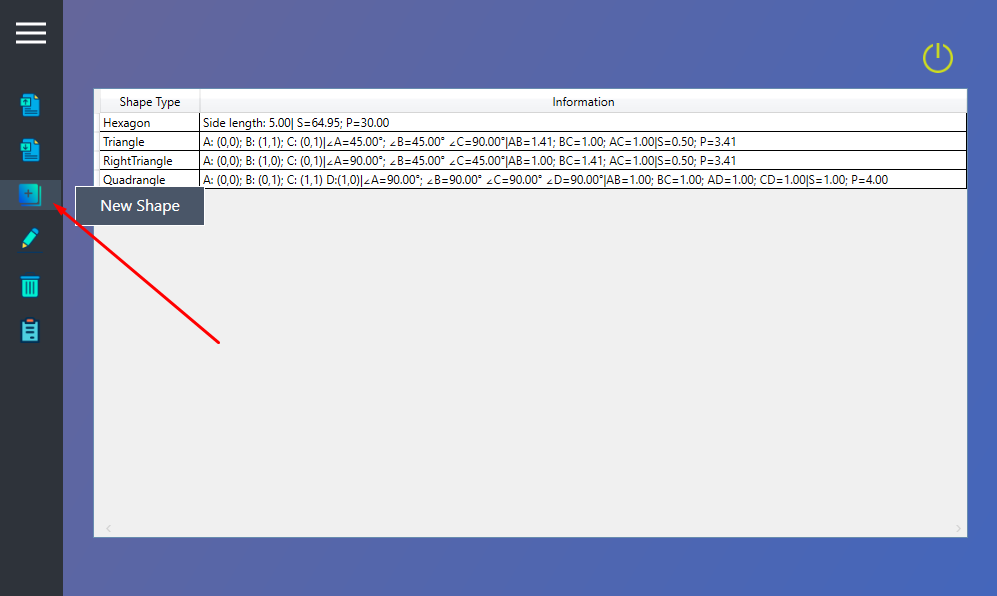


Рис. 6. 7. Кнопка «New Shape»



Рис. 6. 8. Вибір типу фігури зі списку.



Рис. 6. 9. Додавання трикутника в список.

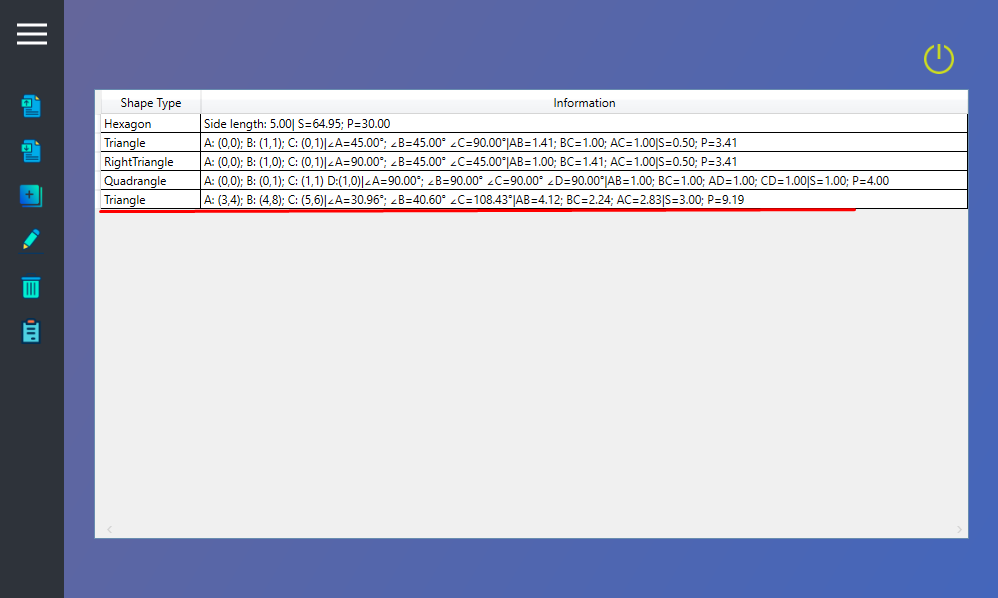


Рис. 6. 10. Нова фігура з’явилась в таблиці.

* Для редагування існуючої фігури потрібно в меню натиснути на кнопку «Edit Shape» (четверта кнопка в меню), з’явиться форма редагування, зі списку потрібно вибрати фігуру, яку хочете редагувати, внести зміни і натиснути на кнопку «Update».

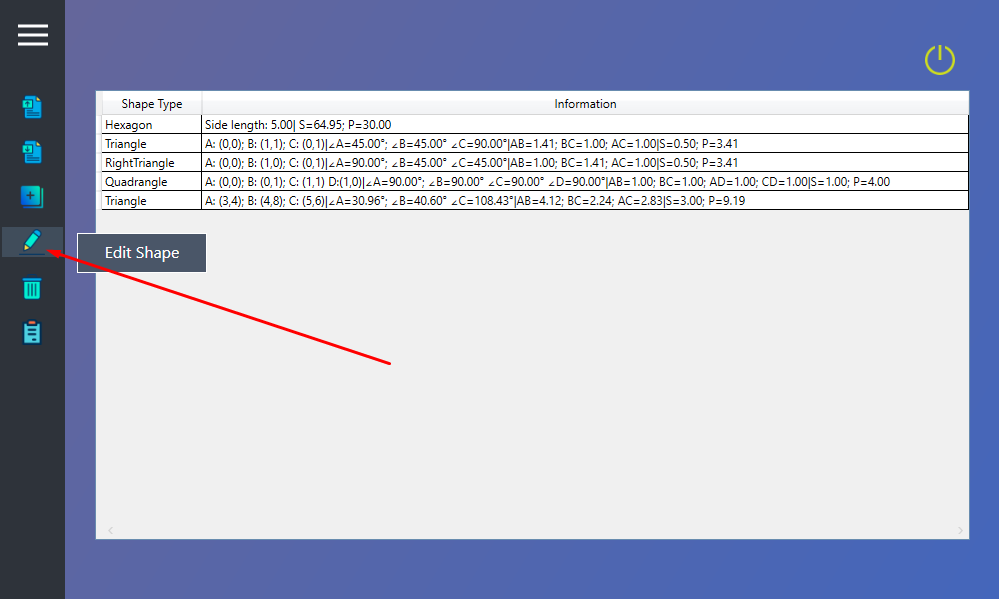


Рис. 6. 11. Кнопка «Edit Shape»

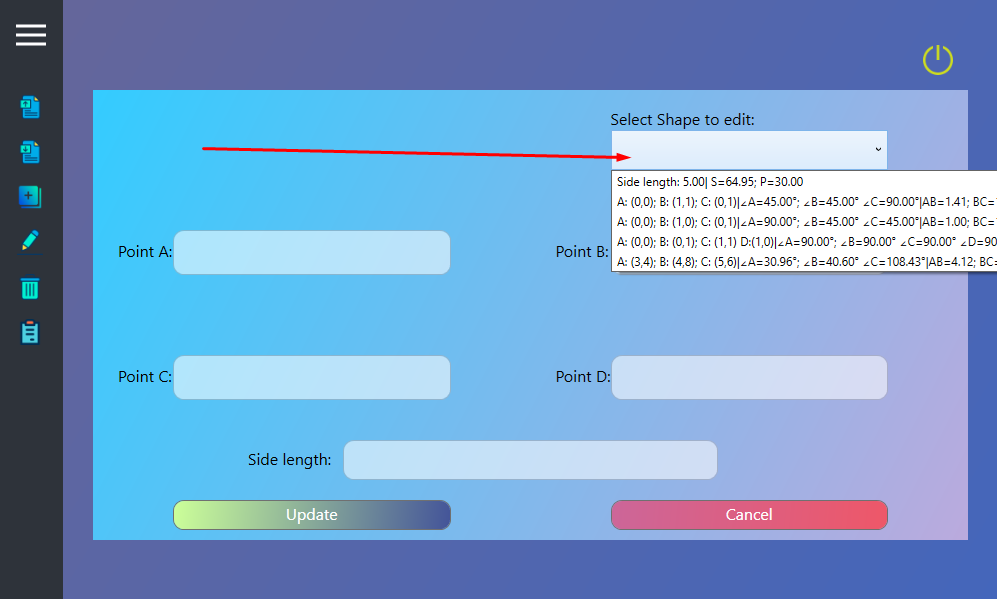


Рис. 6. 12. Вибір фігури для редагування зі списку.

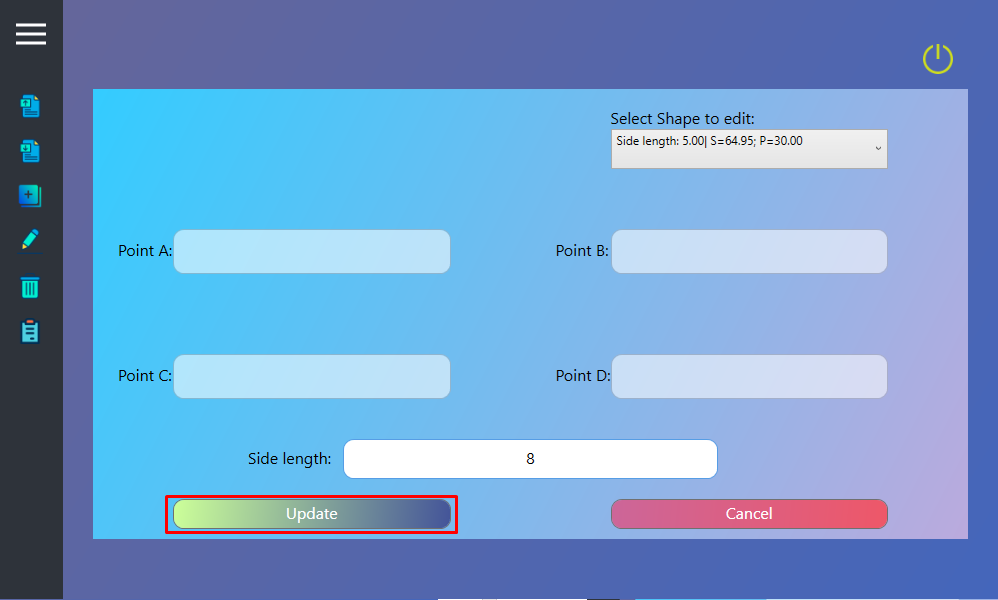


Рис. 6. 13. Збереження змін.

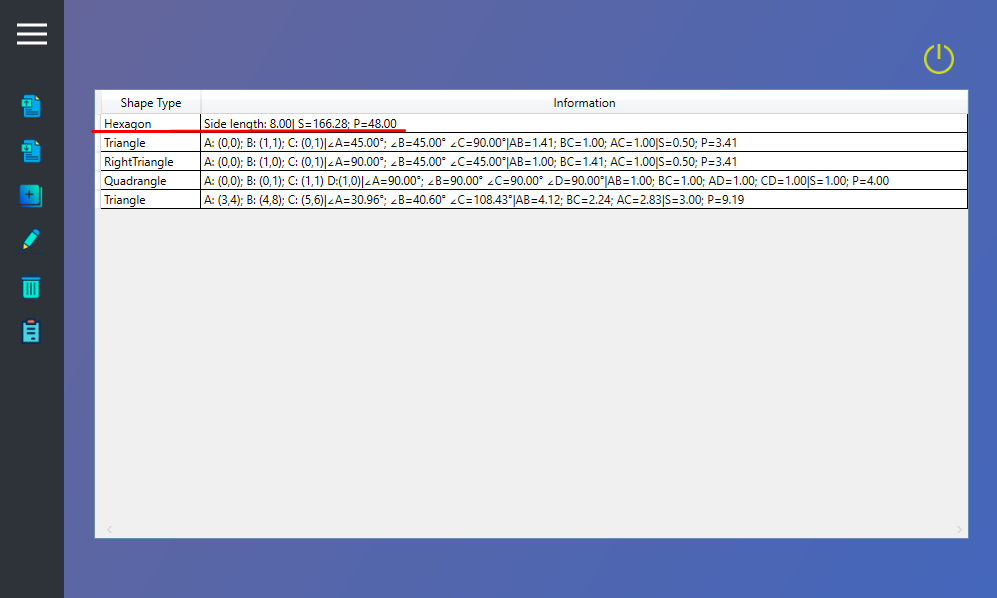


Рис. 6. 14. Зміни відобразились в таблиці.

* Для видалення фігури зі списку потрібно натиснути на кнопку «Remove Shape» (п’ята кнопка в меню), з’явиться форма, де зі списку потрібно вибрати фігуру, яку хочете видалити і натиснути на копку «Remove».

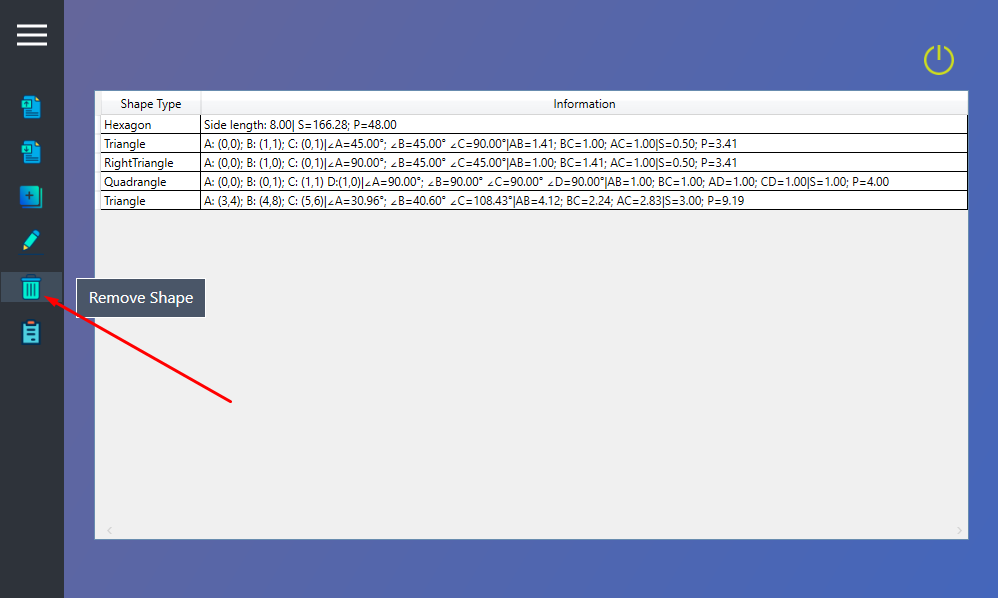


Рис. 6. 15. Кнопка «Remove Shape»

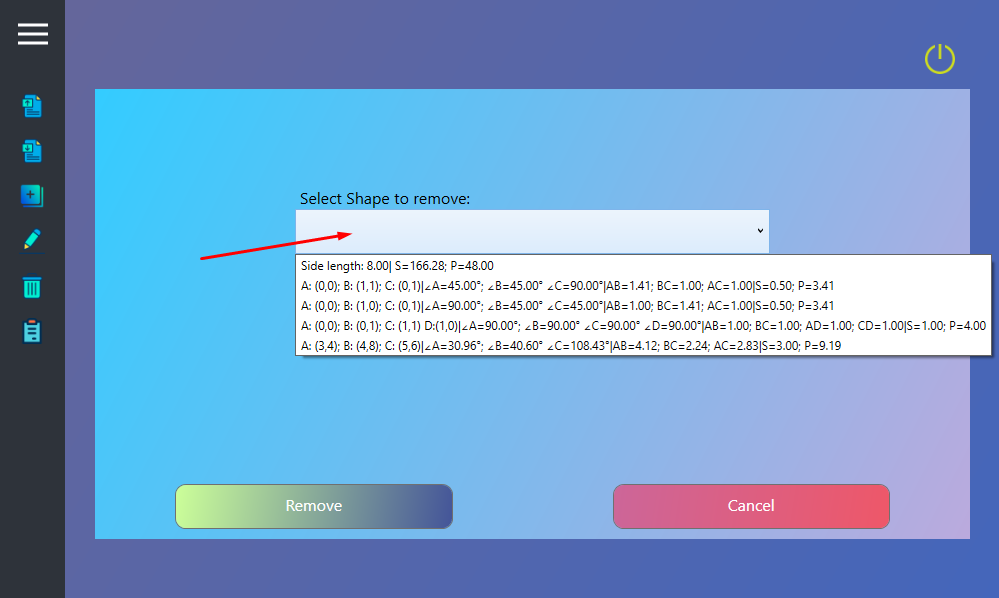


Рис. 6. 16. Вибір фігури для видалення зі списку.

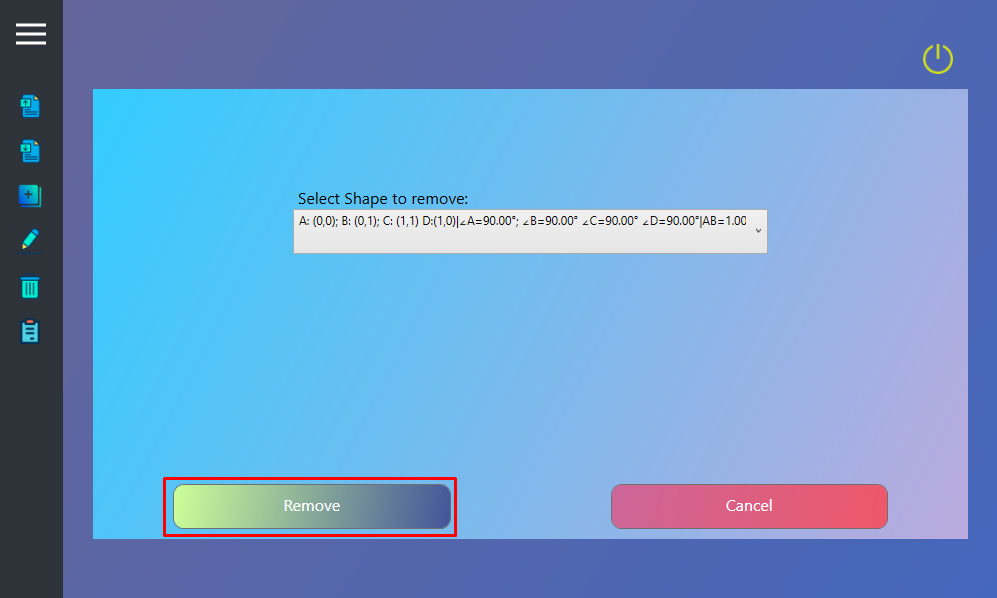


Рис. 6. 16. Видалення фігури.

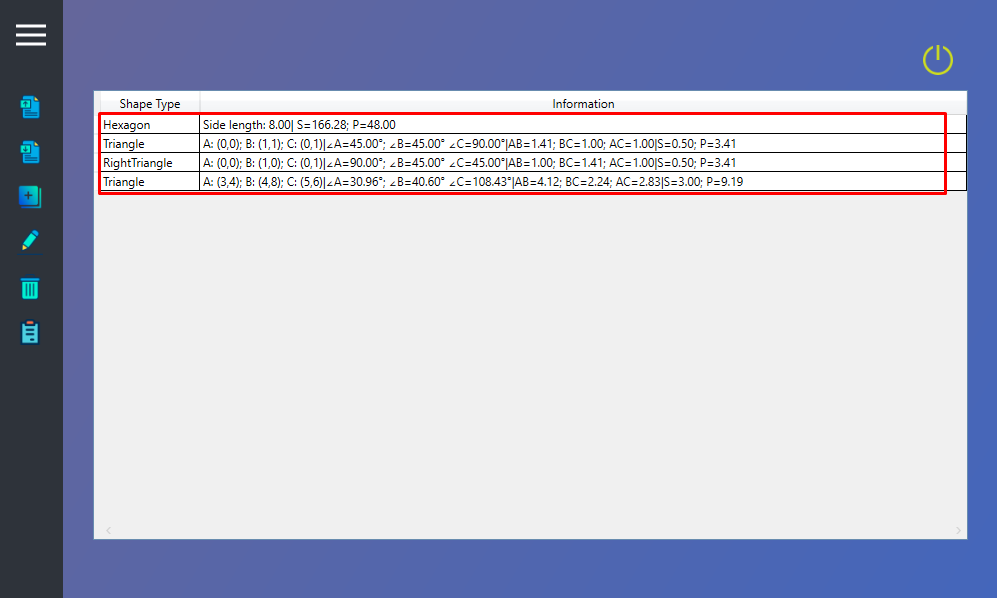


Рис. 6. 17. Фігура зникла зі списку.

* Для збереження історії фігур потрібно в меню натиснути на кнопку «Save history», відкриється файловий провідник, де треба вибрати файл із розширенням txt і натиснути на кнопку «Save».

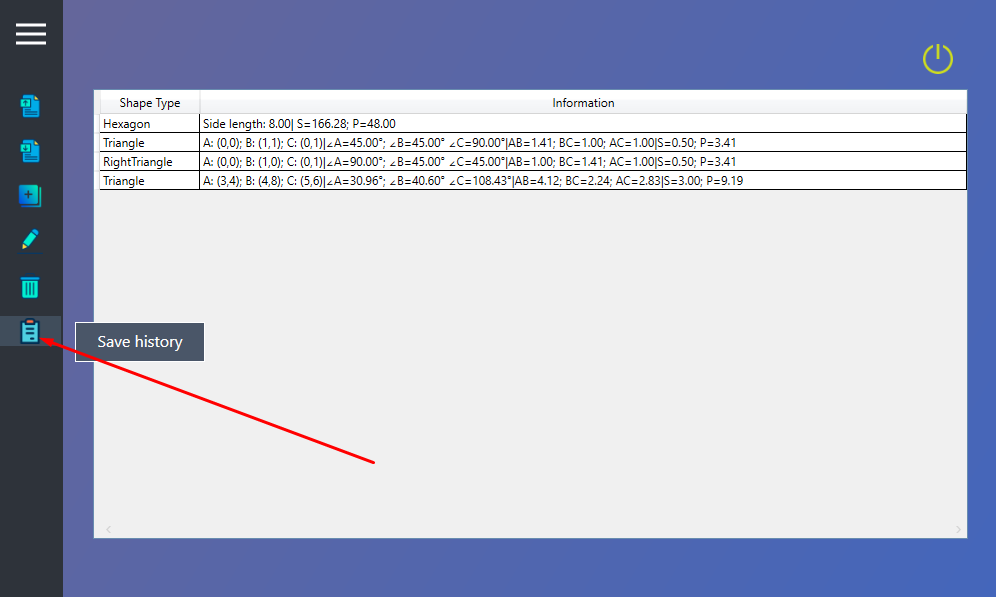


Рис. 6. 18. Кнопка «Save history»

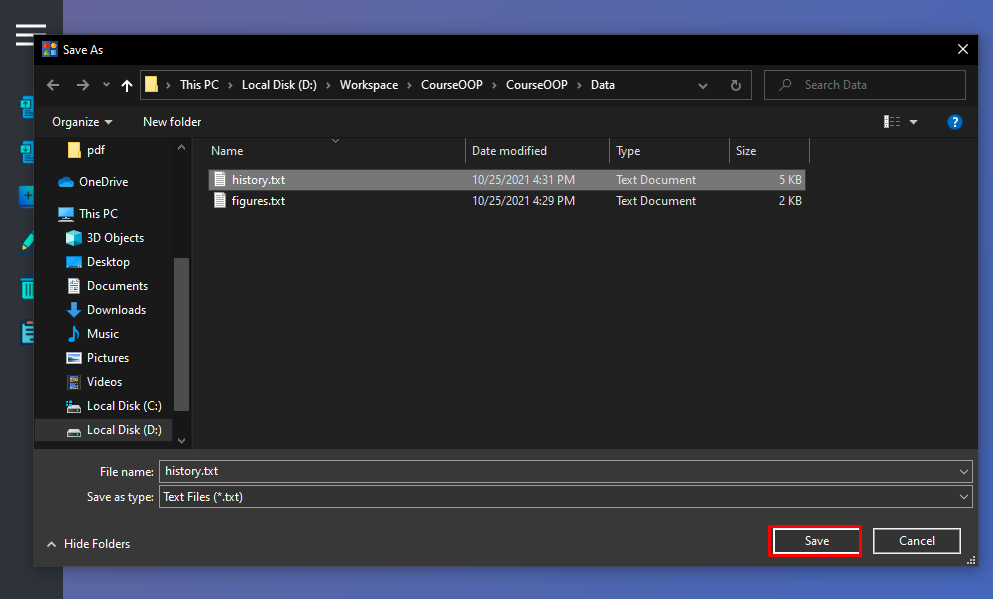


Рис. 6. 19. Вибір файлу для запису історії фігур.

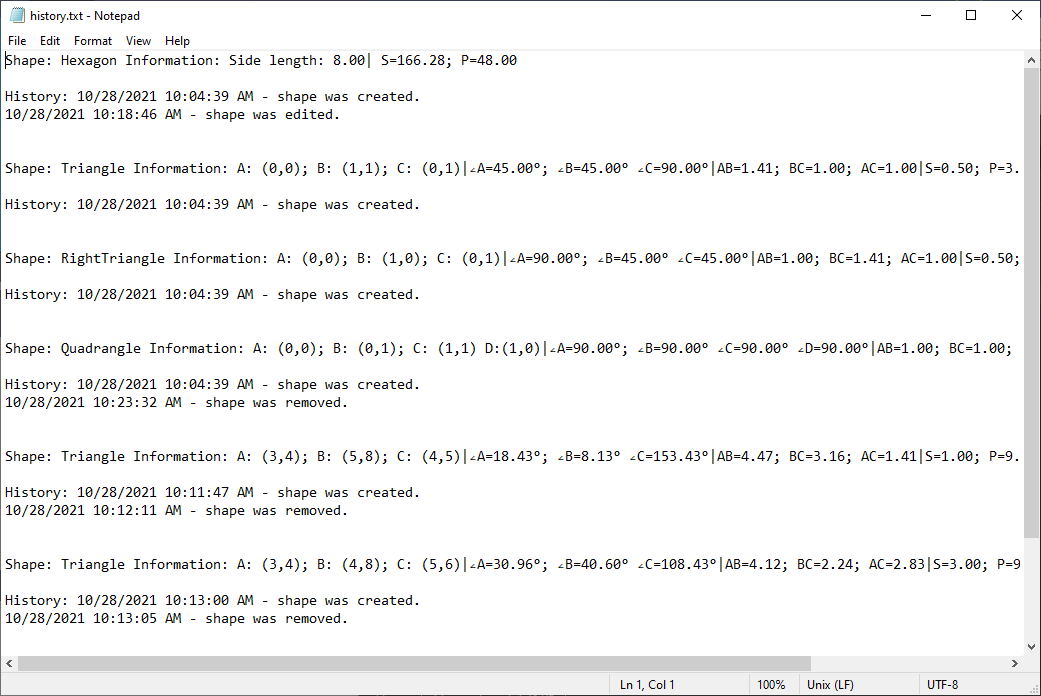


Рис. 6. 20. Історія фігур у файлі.

Приклади вхідних даних:

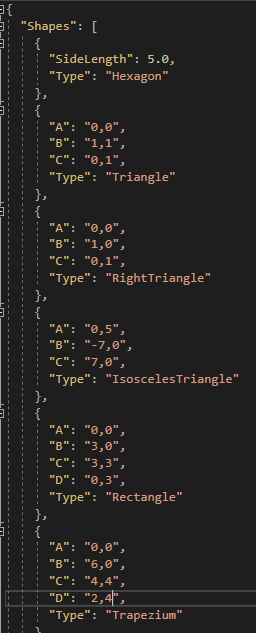


Рис. 6. 21. Приклад вхідного файлу з розширенням json.

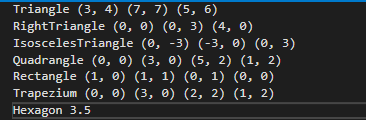


Рис. 6. 22. Приклад вхідного файлу з розширенням txt.

# 

# **Виняткові ситуації**

В програмі передбачено такі виняткові ситуації:

1. Некоректний формат запису фігури у файлі.

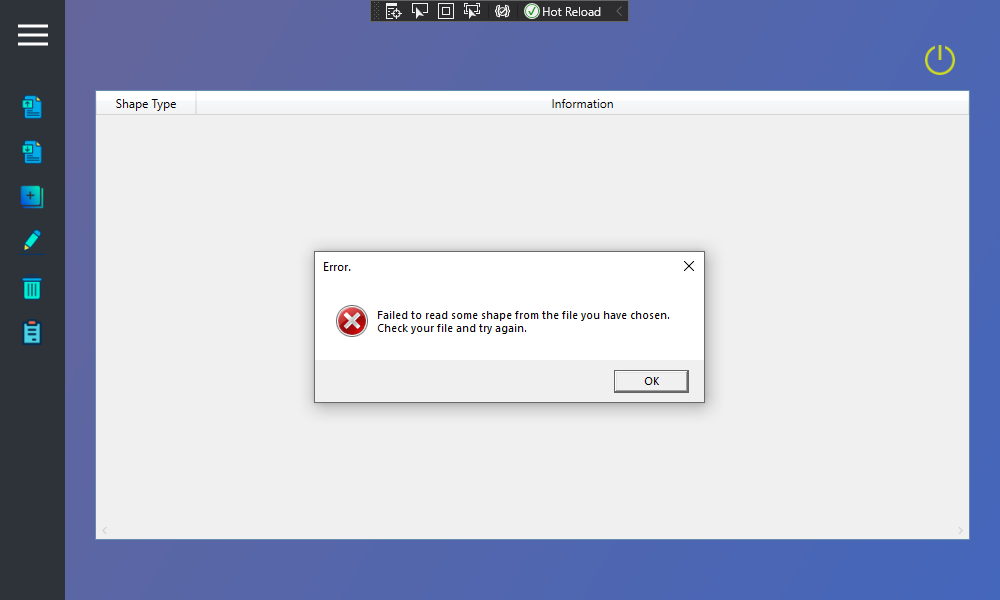


Рис. 7. 1. Помилка: не вдалось прочитати фігуру у файлі.

1. Спроба зчитати фігуру з файлу, тип якої наразі не підтримується програмою.

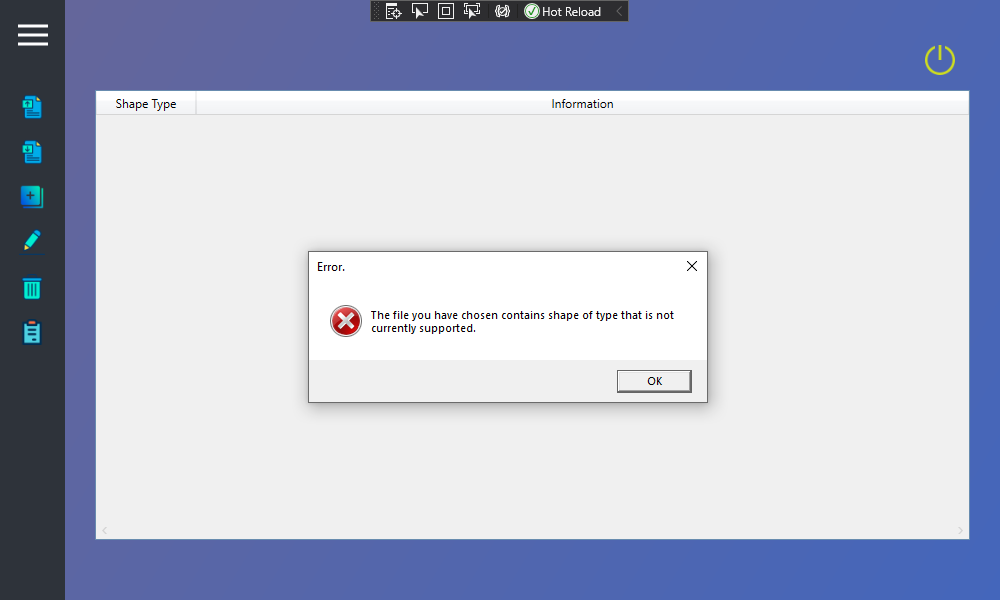


Рис. 7. 2. Помилка: тип фігури, який не підтримується.

1. Некоректне значення поля фігури у файлі.

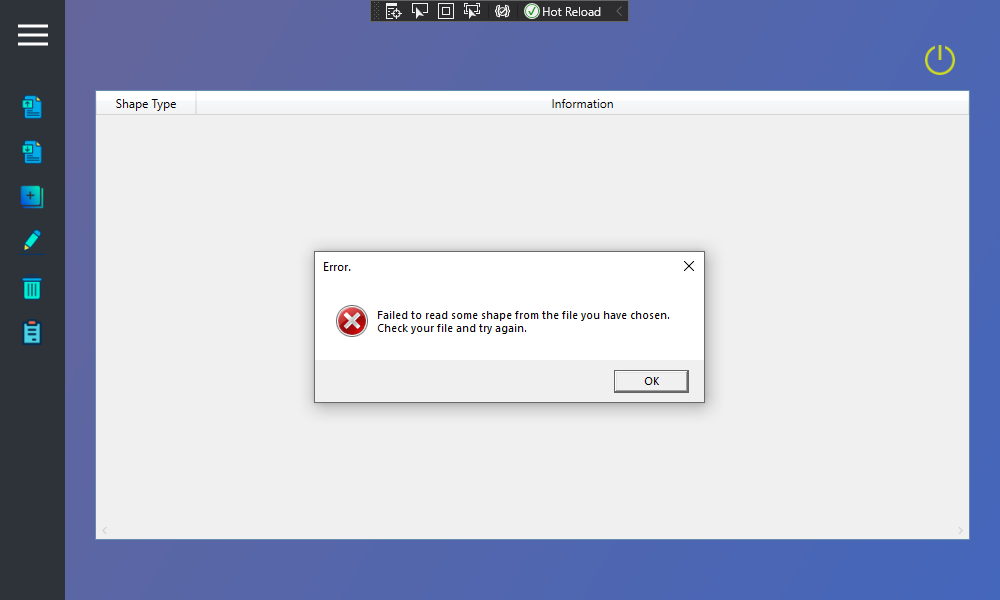


Рис. 7. 3. Помилка: неправильна фігура у файлі.

1. Некоректне значення поля фігури при додаванні або спробі змінити існуючу фігуру.

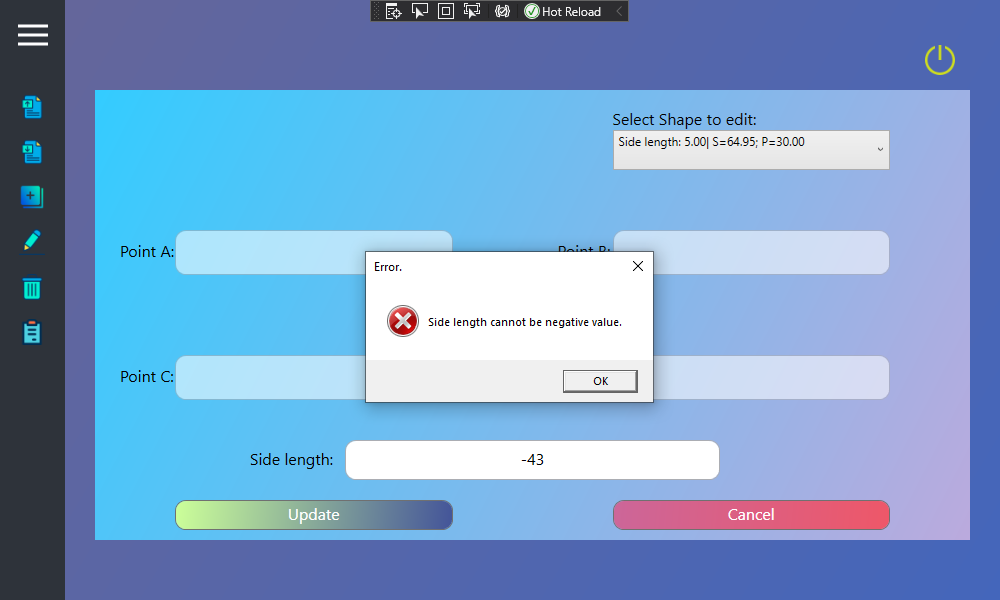


Рис. 7. 4. Помилка: неправильне значення поля «Довжина сторони»

1. Видалення фігури або зміна існуючої фігури в пустому списку.

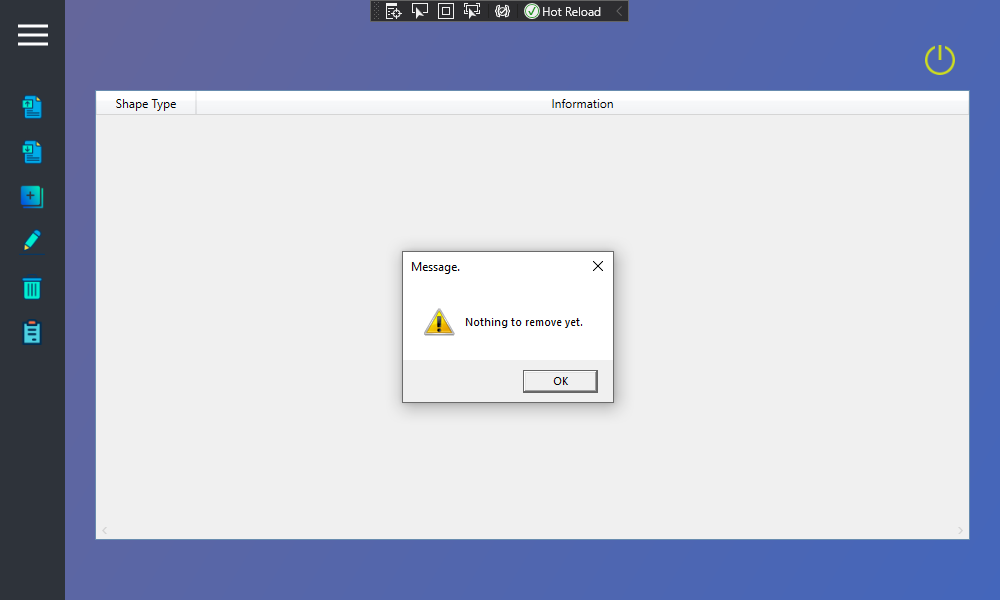
****

Рис. 7. 5. Помилка: не можна видалити фігуру з пустого списку.

# **Висновки**

В ході виконання даної роботи я розробив програму ShapesApp, яка дозволяє керувати плоскими математичними фігурами: Трикутник, Чотирикутник, Шестикутник. Програма автоматично обраховує периметр і площу фігури і відображає на екрані. Цю програму я розробив, використовуючи інтегроване середовище розробки Microsoft Visual Studio 2019 на мові C# із використанням технології WPF. Програму можна підтримувати у майбутньому, додаючи нові типи фігур, розширюючи функціональність та додавши більше інтерактивності.

Дану програму я писав, дотримуючи об’єктно-орієнтованого підходу. Дотримувався основних принципів ООП: інкапсуляція, поліморфізм та наслідування. Особливо в моїй програмі видно застосування наслідування, оскільки я утворив ієрархію класів фігур.

# **Список використаної літератури**

1. <https://docs.microsoft.com/uk-ua/dotnet/csharp/> - офіційна документація мови програмування C# від Microsoft.
2. Joseph Alhabari, Eric Johannsen «C# 8.0 in a Nutshell» - 2020 р. – 1083 ст.
3. <https://habr.com/ru/hub/csharp/> - форум по мові програмування C#.
4. <https://docs.microsoft.com/en-us/visualstudio/designers/getting-started-with-wpf?view=vs-2019> – документація по WPF від Microsoft.