**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ № 1

по дисциплине: «Объектно-ориентированное проектирование и программирование»

на тему: «Основы синтаксиса объектно-ориентированного языка программирования»

Выполнил студент

группы ИТП-21

Лосев М. А.

Проверил ассистент

Карась О. В.

Гомель 2025

**Цель работы:** Создание консольного приложения с интерфейсом пользователя.

**Задание:** Разработать класс, представляющий собой плоскую фигуру, согласно варианту (рисунок 1).



Рисунок 1 – Вариант задания

Ход выполнения задания

1. В приложении А указан код программы*.*
2. На рисунках 2 показан результат запуска программы.
3. На рисунке 3 представлены пройденные тесты

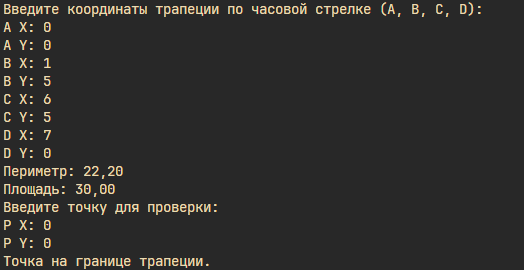


Рисунок 2 – Результат выполнения программы

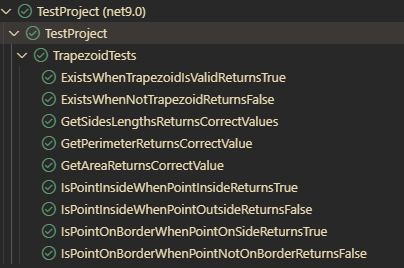


Рисунок 3 – Результат выполнения тестов

В приложении А представлен код программы. В приложении Б представлены полученные *xml* файлы.

**Вывод:** Были изучены методы создания ООП приложений на базе языка *C#* , изучены методы создания библиотек классов и их подключение к основному проекту.

**ПРИЛОЖЕНИЕ А**

(Обязательное)

**Текст программы**

***Program.cs*:**

using ClassLib;

class Program

{

static void Main()

{

Console.WriteLine("Введите координаты трапеции по часовой стрелке (A, B, C, D):");

var a = ReadPoint("A");

var b = ReadPoint("B");

var c = ReadPoint("C");

var d = ReadPoint("D");

var trapezoid = new Trapezoid(a, b, c, d);

if (!trapezoid.Exists())

{

Console.WriteLine("Трапеция с такими координатами не существует.");

return;

}

Console.WriteLine($"Периметр: {trapezoid.GetPerimeter():F2}");

Console.WriteLine($"Площадь: {trapezoid.GetArea():F2}");

Console.WriteLine("Введите точку для проверки:");

var p = ReadPoint("P");

Console.WriteLine(

trapezoid.IsPointOnBorder(p)

? "Точка на границе трапеции."

: trapezoid.IsPointInside(p)

? "Точка внутри трапеции."

: "Точка снаружи трапеции."

);

}

static Point ReadPoint(string name)

{

Console.Write($"{name} X: ");

double x = double.Parse(Console.ReadLine()!);

Console.Write($"{name} Y: ");

double y = double.Parse(Console.ReadLine()!);

return new Point(x, y);

}

***IFigure.cs*:**

namespace ClassLib;

public interface IFigure

{

/// <summary>

/// первая точка

/// </summary>

public Point VertexA { get; set; }

/// <summary>

/// вторая точка

/// </summary>

public Point VertexB { get; set; }

/// <summary>

/// третья точка

/// </summary>

public Point VertexC { get; set; }

/// <summary>

/// четвёртая точка

/// </summary>

public Point VertexD { get; set; }

/// <summary>

/// проверяет может ли существовать фигура

/// </summary>

public bool Exists();

/// <summary>

/// проверят стоит ли точка внутри фигуры

/// </summary>

/// <param name="point">точка для проверки</param>

/// <returns></returns>

public bool IsPointInside(Point point);

/// <summary>

/// проверяет стоит ли точка на границе фигуры

/// </summary>

/// <param name="point">точка для проверки</param>

/// <returns></returns>

public bool IsPointOnBorder(Point point);

/// <summary>

/// возвращает площадь фигуры

/// </summary>

public double GetArea();

/// <summary>

/// возвращает периметр фигуры

/// </summary>

public double GetPerimeter();

/// <summary>

/// возвращает длинны всех сторон фигуры

/// </summary>

public double[] GetSidesLengths();

}

***Trapezoid.cs*:**

namespace ClassLib;

/// <summary>

/// Представляет плоскую фигуру — трапецию на плоскости.

/// </summary>

public class Trapezoid : IFigure

{

/// <inheritdoc/>

public Point VertexA { get; set; }

/// <inheritdoc/>

public Point VertexB { get; set; }

/// <inheritdoc/>

public Point VertexC { get; set; }

/// <inheritdoc/>

public Point VertexD { get; set; }

/// <summary>

/// Создает трапецию с указанными координатами вершин.

/// </summary>

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

{

VertexA = a;

VertexB = b;

VertexC = c;

VertexD = d;

}

/// <inheritdoc/>

public bool Exists()

{

var AB = (X: VertexB.X - VertexA.X, Y: VertexB.Y - VertexA.Y);

var CD = (X: VertexD.X - VertexC.X, Y: VertexD.Y - VertexC.Y);

var BC = (X: VertexC.X - VertexB.X, Y: VertexC.Y - VertexB.Y);

var DA = (X: VertexA.X - VertexD.X, Y: VertexA.Y - VertexD.Y);

return AreVectorsParallel(AB, CD) || AreVectorsParallel(BC, DA);

}

/// <inheritdoc/>

public double[] GetSidesLengths()

{

return

[

CalculateDistance(VertexA, VertexB),

CalculateDistance(VertexB, VertexC),

CalculateDistance(VertexC, VertexD),

CalculateDistance(VertexD, VertexA)

];

}

/// <inheritdoc/>

public double GetPerimeter()

{

double[] sides = GetSidesLengths();

return sides[0] + sides[1] + sides[2] + sides[3];

}

/// <inheritdoc/>

public double GetArea()

{

var AB = (X: VertexB.X - VertexA.X, Y: VertexB.Y - VertexA.Y);

var BC = (X: VertexC.X - VertexB.X, Y: VertexC.Y - VertexB.Y);

var CD = (X: VertexD.X - VertexC.X, Y: VertexD.Y - VertexC.Y);

var DA = (X: VertexA.X - VertexD.X, Y: VertexA.Y - VertexD.Y);

Point baseFirstStart, baseFirstEnd, baseSecondStart, baseSecondEnd;

if (AreVectorsParallel(AB, CD))

{

baseFirstStart = VertexA; baseFirstEnd = VertexB;

baseSecondStart = VertexC; baseSecondEnd = VertexD;

}

else if (AreVectorsParallel(BC, DA))

{

baseFirstStart = VertexB; baseFirstEnd = VertexC;

baseSecondStart = VertexD; baseSecondEnd = VertexA;

}

else

{

throw new InvalidOperationException("Трапеция не имеет параллельных оснований");

}

double a = CalculateDistance(baseFirstStart, baseFirstEnd);

double b = CalculateDistance(baseSecondStart, baseSecondEnd);

// Высота через расстояние от точки до прямой

double dx = baseSecondEnd.X - baseSecondStart.X;

double dy = baseSecondEnd.Y - baseSecondStart.Y;

double h = Math.Abs((dy \* baseFirstStart.X - dx \* baseFirstStart.Y + baseSecondEnd.X \* baseSecondStart.Y - baseSecondEnd.Y \* baseSecondStart.X) / Math.Sqrt(dx \* dx + dy \* dy));

return (a + b) / 2 \* h;

}

/// <inheritdoc/>

public bool IsPointOnBorder(Point point)

{

return IsPointOnLine(VertexA, VertexB, point) ||

IsPointOnLine(VertexB, VertexC, point) ||

IsPointOnLine(VertexC, VertexD, point) ||

IsPointOnLine(VertexD, VertexA, point);

}

/// <inheritdoc/>

public bool IsPointInside(Point point)

{

Point[] vertices = { VertexA, VertexB, VertexC, VertexD };

int count = 0;

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

{

var a = vertices[i];

var b = vertices[(i + 1) % 4];

if ((a.Y > point.Y) != (b.Y > point.Y) && point.X < (b.X - a.X) \* (point.Y - a.Y) / (b.Y - a.Y + 1e-10) + a.X)

{

count++;

}

}

return count % 2 == 1;

}

/// <inheritdoc/>

private bool IsPointOnLine(Point firstPoint, Point secondPoint, Point point)

{

double cross = (point.Y - firstPoint.Y) \* (secondPoint.X - firstPoint.X) - (point.X - firstPoint.X) \* (secondPoint.Y - firstPoint.Y);

if (cross != 0) return false;

double dot = (point.X - firstPoint.X) \* (secondPoint.X - firstPoint.X) + (point.Y - firstPoint.Y) \* (secondPoint.Y - firstPoint.Y);

if (dot < 0) return false;

double lenSq = (secondPoint.X - firstPoint.X) \* (secondPoint.X - firstPoint.X) + (secondPoint.Y - firstPoint.Y) \* (secondPoint.Y - firstPoint.Y);

return dot <= lenSq;

}

/// <summary>

/// Проверяет вектора на параллельность

/// </summary>

private bool AreVectorsParallel((double X, double Y) firstVector, (double X, double Y) secondVector)

{

return (firstVector.X \* secondVector.Y - firstVector.Y \* secondVector.X) == 0;

}

/// <summary>

/// Вычисляет длину стороны между двумя точками.

/// </summary>

private double CalculateDistance(Point firstPoint, Point secondPoint)

{

return Math.Sqrt(Math.Pow(secondPoint.X - firstPoint.X, 2) + Math.Pow(secondPoint.Y - firstPoint.Y, 2));

}

}

}

**ПРИЛОЖЕНИЕ Б**

(Обязательное)

**Структура программы**

***ClassLib.xml***

**<?xml version="1.0"?>**

**<doc>**

**<assembly>**

**<name>ClassLib</name>**

**</assembly>**

**<members>**

**<member name="P:ClassLib.IFigure.VertexA">**

**<summary>**

**первая точка**

**</summary>**

**</member>**

**<member name="P:ClassLib.IFigure.VertexB">**

**<summary>**

**вторая точка**

**</summary>**

**</member>**

**<member name="P:ClassLib.IFigure.VertexC">**

**<summary>**

**третья точка**

**</summary>**

**</member>**

**<member name="P:ClassLib.IFigure.VertexD">**

**<summary>**

**четвёртая точка**

**</summary>**

**</member>**

**<member name="M:ClassLib.IFigure.Exists">**

**<summary>**

**проверяет может ли существовать фигура**

**</summary>**

**</member>**

**<member name="M:ClassLib.IFigure.IsPointInside(ClassLib.Point)">**

**<summary>**

**проверят стоит ли точка внутри фигуры**

**</summary>**

**<param name="point">точка для проверки</param>**

**<returns></returns>**

**</member>**

**<member name="M:ClassLib.IFigure.IsPointOnBorder(ClassLib.Point)">**

**<summary>**

**проверяет стоит ли точка на границе фигуры**

**</summary>**

**<param name="point">точка для проверки</param>**

**<returns></returns>**

**</member>**

**<member name="M:ClassLib.IFigure.GetArea">**

**<summary>**

**возвращает площадь фигуры**

**</summary>**

**</member>**

**<member name="M:ClassLib.IFigure.GetPerimeter">**

**<summary>**

**возвращает периметр фигуры**

**</summary>**

**</member>**

**<member name="M:ClassLib.IFigure.GetSidesLengths">**

**<summary>**

**возвращает длинны всех сторон фигуры**

**</summary>**

**</member>**

**<member name="T:ClassLib.Trapezoid">**

**<summary>**

**Представляет плоскую фигуру — трапецию на плоскости.**

**</summary>**

**</member>**

**<member name="P:ClassLib.Trapezoid.VertexA">**

**<inheritdoc/>**

**</member>**

**<member name="P:ClassLib.Trapezoid.VertexB">**

**<inheritdoc/>**

**</member>**

**<member name="P:ClassLib.Trapezoid.VertexC">**

**<inheritdoc/>**

**</member>**

**<member name="P:ClassLib.Trapezoid.VertexD">**

**<inheritdoc/>**

**</member>**

**<member name="M:ClassLib.Trapezoid.#ctor(ClassLib.Point,ClassLib.Point,ClassLib.Point,ClassLib.Point)">**

**<summary>**

**Создает трапецию с указанными координатами вершин.**

**</summary>**

**</member>**

**<member name="M:ClassLib.Trapezoid.Exists">**

**<inheritdoc/>**

**</member>**

**<member name="M:ClassLib.Trapezoid.GetSidesLengths">**

**<inheritdoc/>**

**</member>**

**<member name="M:ClassLib.Trapezoid.GetPerimeter">**

**<inheritdoc/>**

**</member>**

**<member name="M:ClassLib.Trapezoid.GetArea">**

**<inheritdoc/>**

**</member>**

**<member name="M:ClassLib.Trapezoid.IsPointOnBorder(ClassLib.Point)">**

**<inheritdoc/>**

**</member>**

**<member name="M:ClassLib.Trapezoid.IsPointInside(ClassLib.Point)">**

**<inheritdoc/>**

**</member>**

**<member name="M:ClassLib.Trapezoid.IsPointOnLine(ClassLib.Point,ClassLib.Point,ClassLib.Point)">**

**<inheritdoc/>**

**</member>**

**<member name="M:ClassLib.Trapezoid.AreVectorsParallel(System.ValueTuple{System.Double,System.Double},System.ValueTuple{System.Double,System.Double})">**

**<summary>**

**Проверяет вектора на параллельность**

**</summary>**

**</member>**

**<member name="M:ClassLib.Trapezoid.CalculateDistance(ClassLib.Point,ClassLib.Point)">**

**<summary>**

**Вычисляет длину стороны между двумя точками.**

**</summary>**

**</member>**

**</members>**

**</doc>**