**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ**

**УНИВЕРСИТЕТ ИМЕНИ П.О.СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

специальности 6-05-0611-01 Информационные системы

и технологии (в игровой индустрии)

ЛАБОРАТОРНАЯ РАБОТА №1

по дисциплине: «Объектно-ориентированное проектирование и программирование»

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

Выполнил: студент гр. ИТИ-21

Спиридонов Д.А.

Принял: ассистент

Панарин К.А.

Гомель 2024

# **Задание:**

1. Необходимо разработать класс, представляющий собой плоскую фигуру, согласно варианту (таблица 1).
2. При именовании класса, полей и методов руководствоваться соглашением о наименовании кода ([*https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/inside-a-program/coding-conventions*](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/inside-a-program/coding-conventions)).
3. Фигуру следует задавать координатами вершин.
4. Реализовать методы проверки возможности существования данной фигуры.
5. Реализовать методы вычисления для сторон, площади и периметра.
6. Реализовать методы, проверяющие принадлежность точки, заданной своими координатами на плоскости, фигуре, её границе.
7. Класс должен быть размещён в библиотеке классов.
8. Весь код должен быть снабжён элементами документирования.
9. Сгенерировать XML-файл с документацией по проекту.
10. Создать консольное приложение с интерфейсом пользователя.
11. Подключить созданную библиотеку в консольное приложение.
12. Верифицировать разработанное приложение.
13. Сгенерировать исполняемый файл для последующего переноса на другие компьютеры.

Таблица 1

|  |  |
| --- | --- |
| Вариант | Условие задачи |
| 7 | Параллелограмм |

**Ход работы:**

1. Разработка класса «Параллелограмм»

На рисунке 1 представлен результат создания класса с названием *Parallelogram*.*cs* для проекта *ParallelogramApp*.

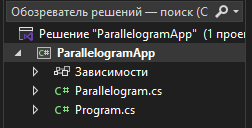


Рисунок 1 – класс *Parallelogram*.*cs*

1. Задание координат вершин параллелограмма

На рисунке 2 представлен фрагмент кода, в котором объявляются переменные *Vertex*1, *Vertex*2, *Vertex*3, *Vertex*4, являющиеся вершинами параллелограмма.

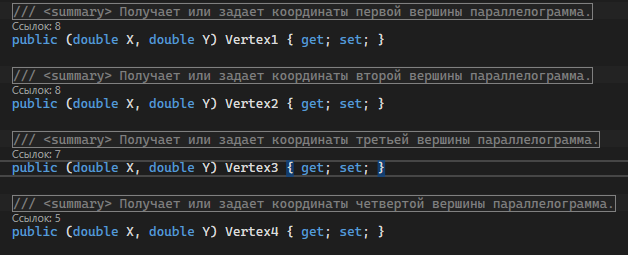


Рисунок 2 – Объявление координат вершин параллелограмма

1. Метод проверки возможности существования данного параллелограмма.

На рисунке 3 представлен фрагмент кода, в котором описывается два метода *IsValid* и *AreParallel*, осуществляющие непосредственно проверку на существование фигуры.

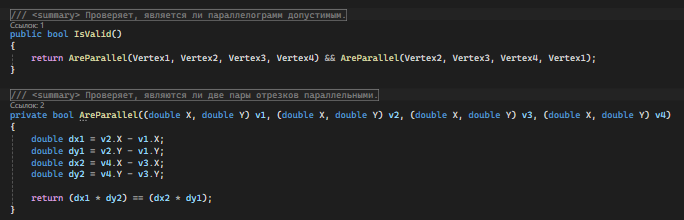


Рисунок 3 – Фрагмент кода, реализующий проверку на существование фигуры

1. Методы вычисления для сторон, площади и периметра.

На рисунке 4 представлен фрагмент кода для вычисления длины отрезка между вершинами, периметра параллелограмма и его площади. Метод *SideLength*() отвечает за вычисление длины отрезка между двумя вершинами, метод *GetPerimeter*() отвечает за вычисление периметра параллелограмма, а метод *GetArea*() отвечает за вычисление его площади.

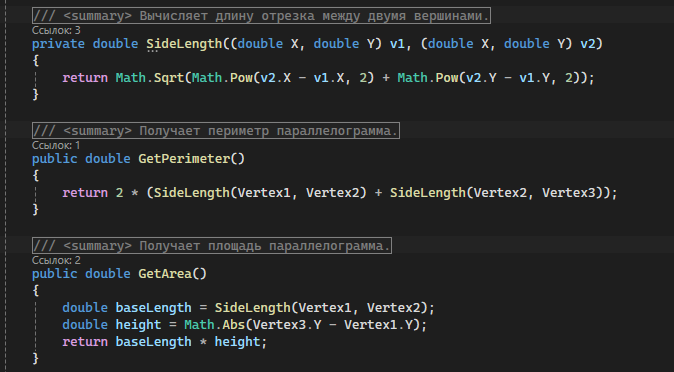


Рисунок 4 – Фрагмент кода для вычисления длины отрезка между вершинами, периметра параллелограмма и его площади

1. Методы, проверяющие принадлежность точки, заданной своими координатами на плоскости, фигуре, её границе.

На рисунке 5 представлен фрагмент кода, в котором описан метод *IsPointInside*(), отвечающий за проверку нахождения данной точки внутри параллелограмма.

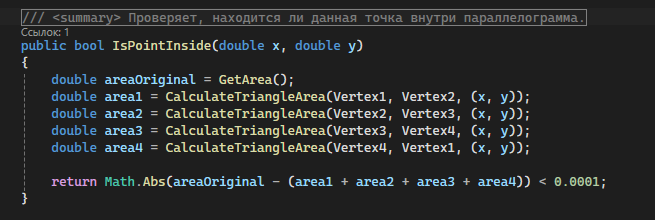


Рисунок 5 – Метод, проверяющий нахождение данной точки внутри параллелограмма

1. Добавление элементов документирования в проект.

На рисунке 6 изображен результат добавления элементов документирования в проект (фрагмент кода).

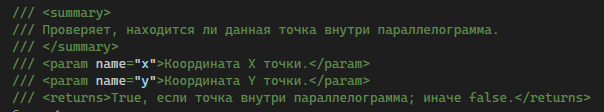


Рисунок 6 – Элементы документирования

Тег <summary> отвечает за описание общего назначения класса или метода. Тег <param> отвечает за описание параметров метода, включая их тип и назначение. Тег <returns> Описание значения, возвращаемого методом. Используется для методов, которые возвращают какое-либо значение. Тег <see cref="TypeName"/> – ссылка на другой элемент кода, что позволяет создать гиперссылку в сгенерированной документации.

Каждое определение класса начинается с комментария, который объясняет, что делает класс. Это помогает пользователям понять назначение класса без необходимости изучать весь код.

1. Генерация *XML*-файла с документацией по проекту.

На рисунке 7 изображен результат генерации *XML*-файла с документацией по проекту и его содержимое.

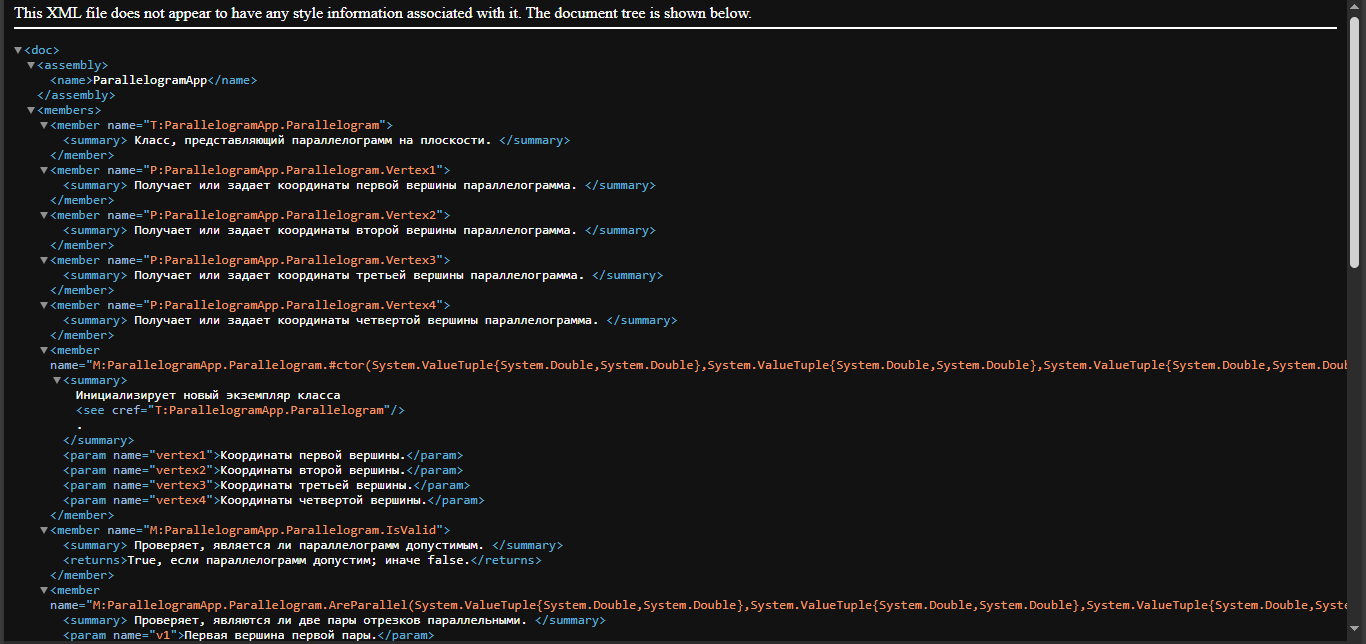


Рисунок 7 – *XML*-файл

1. Консольное приложение с интерфейсом пользователя.

На рисунке 8 изображен вид консольного приложения с интерфейсом пользователя.

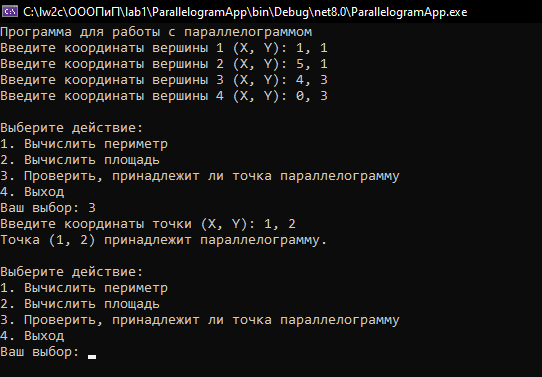


Рисунок 8 – Консольное приложение с интерфейсом

Интерфейс пользователя представлен в виде меню, с которым он может взаимодействовать, выбрав нужные действия (1-3): вычислить периметр, площадь фигуры или проверить принадлежность нужной точки параллелограмму, а также пользователь может выйти из программы, введя число 4.

1. Верификация разработанного приложения.

В таблице 2 представлены исходные данные для тестирования программы.

Таблица 2

|  |  |
| --- | --- |
| Тест | Ожидаемый результат |
| Введите координаты вершины 1 (X, Y): 1, 1  Введите координаты вершины 2 (X, Y): 5, 1  Введите координаты вершины 3 (X, Y): 4, 3  Введите координаты вершины 4 (X, Y): 0, 3  Выберите действие:   1. Вычислить периметр   2. Вычислить площадь  3. Проверить, принадлежит ли точка параллелограмму | Периметр параллелограмма: 12,47213595499958  Площадь параллелограмма: 8  Введите координаты точки (X, Y): 1, 2  Точка (1, 2) принадлежит параллелограмму. |

На рисунке 9 изображен вид консольного приложения с интерфейсом пользователя.

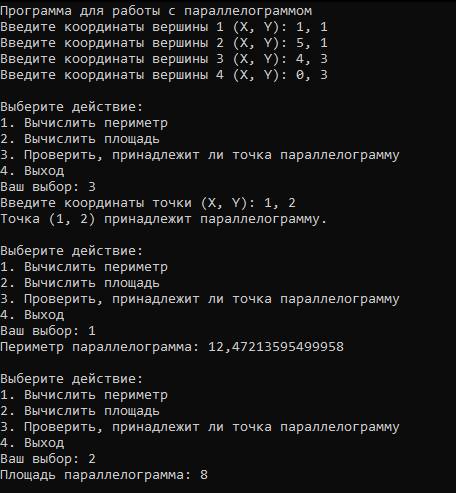


Рисунок 9 – Консольное приложение с интерфейсом

Прогнозируемые результаты в тестах и результаты, полученные после компиляции программы получились одинаковыми, что говорит о правильности написания кода для всех заданных заданий.

1. Создание исполняемого файла для последующего переноса на другие компьютеры.

На рисунке 10 изображено содержимое папки *Release*, в которой находится приложение *ParallelogramApp.exe*, которое является исполняемым файлом созданного проекта

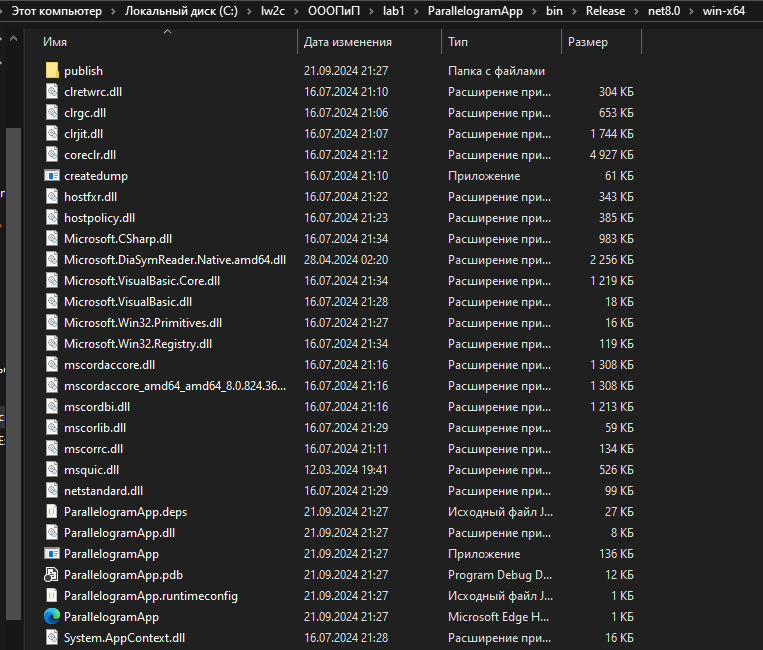


Рисунок 10 – Исполняемый файл *ParallelogramApp.exe*

Данное приложение работает на всех устройствах с установленной операционной системой *Windows* 10 и выше.

Листинг кода программ представлен в приложении А.

**Вывод:** в ходе выполнения лабораторной работы были изучены основы синтаксиса объектно-ориентированного языка программирования, а также было создано и верифицировано консольное приложение с интерфейсом пользователя.

**ПРИЛОЖЕНИЕ А  
(обязательное)  
Листинг кода**

Файл *Parallelogram*.*cs*

using System;

namespace ParallelogramApp

{

/// <summary>

/// Класс, представляющий параллелограмм на плоскости.

/// </summary>

public class Parallelogram

{

/// <summary>

/// Получает или задает координаты первой вершины параллелограмма.

/// </summary>

public (double X, double Y) Vertex1 { get; set; }

/// <summary>

/// Получает или задает координаты второй вершины параллелограмма.

/// </summary>

public (double X, double Y) Vertex2 { get; set; }

/// <summary>

/// Получает или задает координаты третьей вершины параллелограмма.

/// </summary>

public (double X, double Y) Vertex3 { get; set; }

/// <summary>

/// Получает или задает координаты четвертой вершины параллелограмма.

/// </summary>

public (double X, double Y) Vertex4 { get; set; }

/// <summary>

/// Инициализирует новый экземпляр класса <see cref="Parallelogram"/>.

/// </summary>

/// <param name="vertex1">Координаты первой вершины.</param>

/// <param name="vertex2">Координаты второй вершины.</param>

/// <param name="vertex3">Координаты третьей вершины.</param>

/// <param name="vertex4">Координаты четвертой вершины.</param>

public Parallelogram((double X, double Y) vertex1, (double X, double Y) vertex2, (double X, double Y) vertex3, (double X, double Y) vertex4)

{

Vertex1 = vertex1;

Vertex2 = vertex2;

Vertex3 = vertex3;

Vertex4 = vertex4;

}

/// <summary>

/// Проверяет, является ли параллелограмм допустимым.

/// </summary>

/// <returns>True, если параллелограмм допустим; иначе false.</returns>

public bool IsValid()

{

return AreParallel(Vertex1, Vertex2, Vertex3, Vertex4) && AreParallel(Vertex2, Vertex3, Vertex4, Vertex1);

}

/// <summary>

/// Проверяет, являются ли две пары отрезков параллельными.

/// </summary>

/// <param name="v1">Первая вершина первой пары.</param>

/// <param name="v2">Вторая вершина первой пары.</param>

/// <param name="v3">Первая вершина второй пары.</param>

/// <param name="v4">Вторая вершина второй пары.</param>

/// <returns>True, если отрезки параллельны; иначе false.</returns>

private bool AreParallel((double X, double Y) v1, (double X, double Y) v2, (double X, double Y) v3, (double X, double Y) v4)

{

double dx1 = v2.X - v1.X; // Изменение по X для первого отрезка

double dy1 = v2.Y - v1.Y;

double dx2 = v4.X - v3.X; // Изменение по X для второго отрезка

double dy2 = v4.Y - v3.Y;

return (dx1 \* dy2) == (dx2 \* dy1);

}

/// <summary>

/// Вычисляет длину отрезка между двумя вершинами.

/// </summary>

/// <param name="v1">Первая вершина.</param>

/// <param name="v2">Вторая вершина.</param>

/// <returns>Длину отрезка как <see cref="double"/>.</returns>

private double SideLength((double X, double Y) v1, (double X, double Y) v2)

{

return Math.Sqrt(Math.Pow(v2.X - v1.X, 2) + Math.Pow(v2.Y - v1.Y, 2));

}

/// <summary>

/// Получает периметр параллелограмма.

/// </summary>

/// <returns>Периметр параллелограмма как <see cref="double"/>.</returns>

public double GetPerimeter()

{

return 2 \* (SideLength(Vertex1, Vertex2) + SideLength(Vertex2, Vertex3));

}

/// <summary>

/// Получает площадь параллелограмма.

/// </summary>

/// <returns>Площадь параллелограмма как <see cref="double"/>.</returns>

public double GetArea()

{

double baseLength = SideLength(Vertex1, Vertex2);

double height = Math.Abs(Vertex3.Y - Vertex1.Y);

return baseLength \* height;

}

/// <summary>

/// Проверяет, находится ли данная точка внутри параллелограмма.

/// </summary>

/// <param name="x">Координата X точки.</param>

/// <param name="y">Координата Y точки.</param>

/// <returns>True, если точка внутри параллелограмма; иначе false.</returns>

public bool IsPointInside(double x, double y)

{

double areaOriginal = GetArea();

double area1 = CalculateTriangleArea(Vertex1, Vertex2, (x, y));

double area2 = CalculateTriangleArea(Vertex2, Vertex3, (x, y));

double area3 = CalculateTriangleArea(Vertex3, Vertex4, (x, y));

double area4 = CalculateTriangleArea(Vertex4, Vertex1, (x, y));

return Math.Abs(areaOriginal - (area1 + area2 + area3 + area4)) < 0.0001;

}

/// <summary>

/// Вычисляет площадь треугольника по координатам его вершин.

/// </summary>

/// <param name="v1">Первая вершина треугольника.</param>

/// <param name="v2">Вторая вершина треугольника.</param>

/// <param name="v3">Третья вершина треугольника.</param>

/// <returns>Площадь треугольника как <see cref="double"/>.</returns>

private double CalculateTriangleArea((double X, double Y) v1, (double X, double Y) v2, (double X, double Y) v3)

{

return Math.Abs((v1.X \* (v2.Y - v3.Y) + v2.X \* (v3.Y - v1.Y) + v3.X \* (v1.Y - v2.Y)) / 2.0);

}

}

}

Файл *Program*.*cs*

using ParallelogramApp;

class Program

{

/// <summary>

/// Главный метод программы.

/// </summary>

/// <param name="args">Аргументы командной строки.</param>

static void Main(string[] args)

{

Console.WriteLine("Программа для работы с параллелограммом");

var vertex1 = ReadPoint("Введите координаты вершины 1 (X, Y): ");

var vertex2 = ReadPoint("Введите координаты вершины 2 (X, Y): ");

var vertex3 = ReadPoint("Введите координаты вершины 3 (X, Y): ");

var vertex4 = ReadPoint("Введите координаты вершины 4 (X, Y): ");

Parallelogram parallelogram = new Parallelogram(vertex1, vertex2, vertex3, vertex4);

if (!parallelogram.IsValid())

{

Console.WriteLine("Параллелограмм с данными координатами не существует.");

return;

}

int choice;

do

{

Console.WriteLine("\nВыберите действие:");

Console.WriteLine("1. Вычислить периметр");

Console.WriteLine("2. Вычислить площадь");

Console.WriteLine("3. Проверить, принадлежит ли точка параллелограмму");

Console.WriteLine("4. Выход");

Console.Write("Ваш выбор: ");

// Проверка на корректный ввод

while (!int.TryParse(Console.ReadLine(), out choice) || choice < 1 || choice > 4)

{

Console.WriteLine("Некорректный ввод. Пожалуйста, выберите пункт от 1 до 4.");

}

switch (choice)

{

case 1:

Console.WriteLine($"Периметр параллелограмма: {parallelogram.GetPerimeter()}");

break;

case 2:

Console.WriteLine($"Площадь параллелограмма: {parallelogram.GetArea()}");

break;

case 3:

var point = ReadPoint("Введите координаты точки (X, Y): ");

if (parallelogram.IsPointInside(point.X, point.Y))

{

Console.WriteLine($"Точка ({point.X}, {point.Y}) принадлежит параллелограмму.");

}

else

{

Console.WriteLine($"Точка ({point.X}, {point.Y}) не принадлежит параллелограмму.");

}

break;

case 4:

Console.WriteLine("Выход из программы.");

break;

}

} while (choice != 4); // Продолжать, пока пользователь не выберет выход

}

/// <summary>

/// Читает координаты точки из ввода пользователя.

/// </summary>

/// <param name="prompt">Сообщение для запроса ввода.</param>

/// <returns>Координаты точки как кортеж.</returns>

static (double X, double Y) ReadPoint(string prompt)

{

Console.Write(prompt);

var input = Console.ReadLine().Split(',');

if (input.Length != 2 || !double.TryParse(input[0], out double x) || !double.TryParse(input[1], out double y))

{

Console.WriteLine("Неправильный ввод. Попробуйте снова.");

return ReadPoint(prompt);

}

return (x, y);

}

}