**Лабораторная работа №3**

**Задание 1**. Создайте проект, в котором опишите класс для решения задачи Вашего варианта.

Разрабатываемый класс должен содержать следующие элементы: скрытые и открытые поля, конструкторы без параметров и с параметрами (имена некоторых полей должны совпадать с идентификаторами параметров), методы и свойства. Методы и свойства должны обеспечивать непротиворечивый и удобный интерфейс класса. В программе должна выполняться проверка всех разработанных элементов класса, вывод состояния объекта.

9. Описать класс, представляющий четырехугольник. Разработать методы

для определения является ли четырехугольник параллелограммом.

using System;

class Quadrilateral

{

// Поля для хранения координат вершин четырехугольника

private double x1, y1;

private double x2, y2;

private double x3, y3;

private double x4, y4;

// Конструктор с параметрами для инициализации вершин

public Quadrilateral(double x1, double y1, double x2, double y2, double x3, double y3, double x4, double y4)

{

this.x1 = x1;

this.y1 = y1;

this.x2 = x2;

this.y2 = y2;

this.x3 = x3;

this.y3 = y3;

this.x4 = x4;

this.y4 = y4;

}

// Метод для определения, является ли четырехугольник параллелограммом

public bool IsParallelogram()

{

// Проверяем, что противоположные стороны параллельны

double slope1 = (y2 - y1) / (x2 - x1);

double slope2 = (y4 - y3) / (x4 - x3);

return Math.Abs(slope1 - slope2) < 0.0001;

}

// Метод для ввода координат вершин

public static Quadrilateral InputQuadrilateral()

{

Console.WriteLine("Введите координаты вершин четырехугольника:");

Console.Write("x1: ");

double x1 = double.Parse(Console.ReadLine());

Console.Write("y1: ");

double y1 = double.Parse(Console.ReadLine());

Console.Write("x2: ");

double x2 = double.Parse(Console.ReadLine());

Console.Write("y2: ");

double y2 = double.Parse(Console.ReadLine());

Console.Write("x3: ");

double x3 = double.Parse(Console.ReadLine());

Console.Write("y3: ");

double y3 = double.Parse(Console.ReadLine());

Console.Write("x4: ");

double x4 = double.Parse(Console.ReadLine());

Console.Write("y4: ");

double y4 = double.Parse(Console.ReadLine());

return new Quadrilateral(x1, y1, x2, y2, x3, y3, x4, y4);

}

}

class Program

{

static void Main()

{

// Вводим координаты вершин четырехугольника

Quadrilateral quad = Quadrilateral.InputQuadrilateral();

// Проверяем, является ли он параллелограммом

if (quad.IsParallelogram())

{

Console.WriteLine("Четырехугольник является параллелограммом.");

}

else

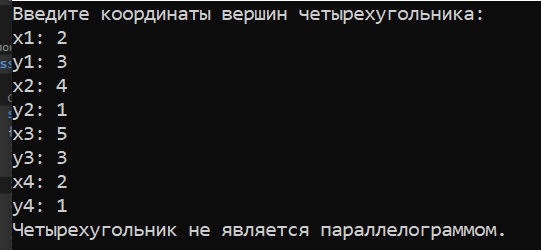
{

Console.WriteLine("Четырехугольник не является параллелограммом.");

}

}

}



**Задание 2**. Включите в проект Задания 1обработку исключений.

using System;

class Quadrilateral

{

// Поля для хранения координат вершин четырехугольника

private double x1, y1;

private double x2, y2;

private double x3, y3;

private double x4, y4;

// Конструктор с параметрами для инициализации вершин

public Quadrilateral(double x1, double y1, double x2, double y2, double x3, double y3, double x4, double y4)

{

this.x1 = x1;

this.y1 = y1;

this.x2 = x2;

this.y2 = y2;

this.x3 = x3;

this.y3 = y3;

this.x4 = x4;

this.y4 = y4;

}

// Метод для определения, является ли четырехугольник параллелограммом

public bool IsParallelogram()

{

// Проверяем, что противоположные стороны параллельны

double slope1 = (y2 - y1) / (x2 - x1);

double slope2 = (y4 - y3) / (x4 - x3);

return Math.Abs(slope1 - slope2) < 0.0001;

}

// Метод для ввода координат вершин

// Метод для ввода координат вершин

public static Quadrilateral InputQuadrilateral()

{

try

{

Console.WriteLine("Введите координаты вершин четырехугольника:");

Console.Write("x1: ");

double x1 = double.Parse(Console.ReadLine());

Console.Write("y1: ");

double y1 = double.Parse(Console.ReadLine());

Console.Write("x2: ");

double x2 = double.Parse(Console.ReadLine());

Console.Write("y2: ");

double y2 = double.Parse(Console.ReadLine());

Console.Write("x3: ");

double x3 = double.Parse(Console.ReadLine());

Console.Write("y3: ");

double y3 = double.Parse(Console.ReadLine());

Console.Write("x4: ");

double x4 = double.Parse(Console.ReadLine());

Console.Write("y4: ");

double y4 = double.Parse(Console.ReadLine());

return new Quadrilateral(x1, y1, x2, y2, x3, y3, x4, y4);

}

catch (FormatException ex)

{

Console.WriteLine("Ошибка формата ввода: " + ex.Message);

return null; // Возвращаем null в случае ошибки

}

}

}

class Program

{

static void Main()

{

// Вводим координаты вершин четырехугольника

Quadrilateral quad = Quadrilateral.InputQuadrilateral();

// Проверяем, является ли он параллелограммом

if (quad.IsParallelogram())

{

Console.WriteLine("Четырехугольник является параллелограммом.");

}

else

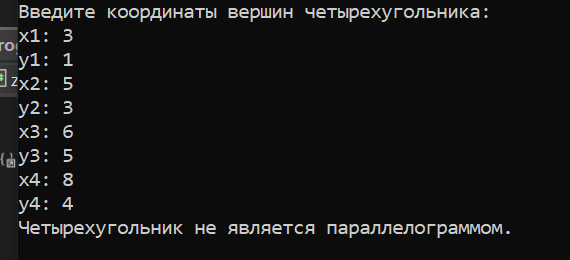
{

Console.WriteLine("Четырехугольник не является параллелограммом.");

}

}

}



Контрольные вопросы:

1. Как описываются классы в С#?

Классы в C# описываются с использованием ключевого слова class, за которым следует имя класса и тело класса в фигурных скобках. Пример описания класса:

public class MyClass

{

// Поля, свойства, методы и другие члены класса

}

2. Что относится к членам класса?

Члены класса включают в себя поля (variables), свойства (properties), методы (methods), события (events), индексаторы (indexers) и конструкторы (constructors). Эти члены определяют состояние и поведение объектов, созданных на основе класса.

3. Что такое статические члены класса?

Статические члены класса относятся к классу в целом, а не к конкретным экземплярам класса. Они могут быть вызваны без создания объекта класса и обычно используются для хранения общей информации или выполнения операций, не зависящих от конкретного состояния объекта. Статические члены класса объявляются с использованием ключевого слова static.

4. Данные: поля и константы.

В классах C# данные могут быть представлены с помощью полей и констант. Поля (fields) представляют переменные, которые хранят состояние объекта класса, а константы (constants) представляют значения, которые остаются постоянными на протяжении выполнения программы.

5. Спецификаторы полей и констант класса.

Спецификаторы полей класса могут включать в себя:

* public: поле доступно из любого места в программе.
* private: поле доступно только внутри класса.
* protected: поле доступно внутри класса и его производных классов.
* internal: поле доступно внутри сборки.
* protected internal: поле доступно внутри сборки и его производных классов.

Спецификаторы констант класса:

* const: определяет константу времени компиляции.
* readonly: определяет константу времени выполнения, которая может быть установлена только в конструкторе класса или при объявлении

6. Как передаются параметры в методы?

Параметры передаются в методы путем указания их типов и имен в списке параметров метода в круглых скобках при объявлении метода. При вызове метода значения аргументов передаются в соответствии с порядком параметров.

7. Для чего предназначен параметр params?

Параметр params в C# позволяет методу принимать переменное количество аргументов одного типа. При использовании параметра params можно передавать любое количество аргументов указанного типа, и компилятор автоматически упаковывает их в массив.

8. Что представляет собой конструктор? Для чего он используется?

Конструктор в C# является специальным методом класса, который вызывается при создании нового объекта этого класса. Он используется для инициализации объекта, установки начальных значений полей, выполнения необходимых действий при создании объекта.

9. Какие бывают конструкторы?

В C# существуют следующие типы конструкторов:

* + Параметризованный конструктор: принимает параметры для инициализации объекта.
  + Конструктор по умолчанию: не принимает параметры и используется, если не определены другие конструкторы.
  + Конструктор копирования: создает новый объект на основе существующего объекта.

10.Может ли класс не иметь конструктора?

Да, класс может не иметь явно определенных конструкторов. В этом случае компилятор автоматически создает конструктор по умолчанию (без параметров), если в классе не определены другие конструкторы.

11.Для чего предназначена система сбора мусора?

Система сбора мусора в C# предназначена для автоматического управления памятью и освобождения ресурсов, занимаемых объектами, когда они больше не нужны. Сборщик мусора отслеживает объекты, на которые нет ссылок из программы, и освобождает память, занимаемую этими объектами, чтобы предотвратить утечку памяти и повысить производительность программы.