

Санкт-Петербургский национальный исследовательский университет
информационных технологий, механики и оптики

Факультет инфокоммуникационных технологий

Направление подготовки 11.03.02

Лабораторная работа №4

Создание и использование методов

Выполнил:

Доценников Никита Андреевич

Группа: К3221

Проверил:

Иванов Сергей Евгеньевич

Санкт-Петербург

2025

Цель работы:

Изучить и приобрести навыки работы с методами класса.

Упражнение 1. Использование параметров в методах, возвращающих значения.

Я создал класс `Utils` и определил в нем метод `Greater`, который принимает 2 целочисленных параметра и возвращает больший из них.

Тело метода:

```
public static int Greater(int a, int b)
{
    if (a > b)
        return a;
    else
        return b;
}
```

Тест:

```
static void Main(string[] args)
{
    int x, y;

    Console.WriteLine("Введите первое число: ");
    x = int.Parse(Console.ReadLine());
    Console.WriteLine("Введите второе число: ");
    y = int.Parse(Console.ReadLine());

    int greater = Utils.Greater(x, y);

    Console.WriteLine("Большим из чисел {0} и {1} является {2}", x, y, greater);
}
```

Пример:

```
Введите первое число:
12
Введите второе число:
21
Большим из чисел 12 и 21 является 21
```

Упражнение 2. Использование в методах параметров, передаваемых по ссылке

Я создал метод `Swap`, который меняет местами значения параметров. При этом использовались параметры, передаваемые по ссылке:

```
public static void Swap(ref int a, ref int b)
{
    int temp = a;
    a = b;
    b = temp;
}
```

Тест:

```
Console.WriteLine("До swap: \t" + x + " " + y);
Utils.Swap(ref x, ref y);
Console.WriteLine("После swap: \t" + x + " " + y);
```

Пример:

```
До swap:      12 21
После swap:   21 12
```

Упражнение 3. Использование возвращаемых параметров в методах.

В этом упражнении я создал метод `Factorial`, принимающий целочисленную переменную и рассчитывающий ее факториал по итерационному алгоритму.

```

public static bool Factorial(int n, out int answer)
{
    int k;
    int f = 1;
    bool ok = true;

    try
    {
        checked
        {
            for (k = 2; k <= n; ++k)
            {
                f = f * k;
            }
        }
    }
    catch (Exception)
    {
        f = 0;
        ok = false;
    }

    answer = f;
    return ok;
}

```

Тест:

```

int f;
bool ok;

Console.WriteLine("Number for factorial:");
x = int.Parse(Console.ReadLine());

ok = Utils.Factorial(x, out f);

if (ok)
    Console.WriteLine("Factorial(" + x + ") = " + f);
else
    Console.WriteLine("Cannot compute this factorial");

```

Пример:

Number for factorial:
4
Factorial(4) = 24

Упражнение 4. Расчет площади треугольника с помощью метода.

В этом упражнении я создал класс `Operation` который умеет считать площадь треугольника, проверять треугольник на существование и считать площадь для равностороннего треугольника.

```
using System;

public class Operation
{
    public static double TriangleArea(double a, double b,
double c)
    {
        double p = (a + b + c) / 2;
        bool exists = CheckTriangle(a, b, c);
        if (!exists)
            return -1;

        return Math.Sqrt(p * (p - a) * (p - b) * (p - c));
    }

    public static double TriangleArea(double a)
    {
        return TriangleArea(a, a, a);
    }

    public static bool CheckTriangle(double a, double b,
double c)
    {
        if (a <= 0 || b <= 0 || c <= 0) return false;

        double maxSide = Math.Max(Math.Max(a, b), Math.Max(b,
c));
    }
}
```

```

        if (maxSide >= a + b + c - maxSide) return false;

        return true;
    }

    public static int SolveQuadratic(int a, int b, int c, out
double x1, out double x2)
    {
        double d = b * b - 4 * a * c;
        if (d > 0)
        {
            x1 = (-b + Math.Sqrt(d)) / (2 * a);
            x2 = (-b - Math.Sqrt(d)) / (2 * a);
            return 1;
        }
        else if (d == 0)
        {
            x1 = x2 = -b / (2 * a);
            return 0;
        }
        else
        {
            x1 = x2 = -1;
            return -1;
        }
    }
}

```

Тесты:

```

Console.WriteLine("choose the type of the triangle. type 1 for
equilateral else 2: ");
int triangleType = int.Parse(Console.ReadLine());

switch (triangleType)
{

    case 1:
        Console.WriteLine("input side length: ");
        double a = double.Parse(Console.ReadLine());
        Console.WriteLine("area = {0}",

```

```

Operation.TriangleArea(a));
    break;
case 2:
    Console.WriteLine("input first side length: ");
    double sideA = double.Parse(Console.ReadLine());
    Console.WriteLine("input second side length: ");
    double sideB = double.Parse(Console.ReadLine());
    Console.WriteLine("input third side length: ");
    double sideC = double.Parse(Console.ReadLine());
    Console.WriteLine("area = {0}",
Operation.TriangleArea(sideA, sideB, sideC));
    break;
default:
    Console.WriteLine("invalid triangle type");
    break;
}

```

Пример:

```

choose the type of the triangle. type 1 for equilateral else 2:
1
input side length:
3
area = 3.897114317029974

```

Упражнение 5. Вычисление корней квадратного уравнения.

В этом упражнении я реализовал метод вычисления квадратного уравнения.

```

public static int SolveQuadratic(int a, int b, int c, out
double x1, out double x2)
{
    double d = b * b - 4 * a * c;
    if (d > 0)
    {
        x1 = (-b + Math.Sqrt(d)) / (2 * a);
        x2 = (-b - Math.Sqrt(d)) / (2 * a);
        return 1;
    }
    else if (d == 0)
    {

```

```

        x1 = x2 = -b / (2 * a);
        return 0;
    }
    else
    {
        x1 = x2 = -1;
        return -1;
    }
}

```

Тесты:

```

Console.WriteLine("input coefficient a: ");
int coef_a = int.Parse(Console.ReadLine());
Console.WriteLine("input coefficient b: ");
int coef_b = int.Parse(Console.ReadLine());
Console.WriteLine("input coefficient c: ");
int coef_c = int.Parse(Console.ReadLine());

double x1, x2;
int res = Operation.SolveQuadratic(coef_a, coef_b, coef_c, out
x1, out x2);

if (res == 1)
{
    Console.WriteLine("Корни уравнения с коэффициентами a =
результат, b = результат, c = результат равны x1 = {0}, x2 =
{1}." , x1, x2);
}
else if (res == 0)
{
    Console.WriteLine("Корень уравнения с коэффициентами a =
результат, b = результат, c = результат равны один x1 = x2 =
{0}" , x1);
}
else
{
    Console.WriteLine("Корней уравнения с коэффициентами a =
результат, b = результат, c = результат нет.");
}

```

Пример:

```
input coefficient a:
1
input coefficient b:
-2
input coefficient c:
1
Корень уравнения с коэффициентами  $a = 1$ ,  $b = -2$ ,  $c = 1$  равны один  $x_1 =$ 
 $x_2 = 1$ 
```

```
input coefficient a:
1
input coefficient b:
1
input coefficient c:
1
Корней уравнения с коэффициентами  $a = 1$ ,  $b = 1$ ,  $c = 1$  нет.
```

```
input coefficient a:
1
input coefficient b:
-5
input coefficient c:
6
Корни уравнения с коэффициентами  $a = 1$ ,  $b = -5$ ,  $c = 6$  равны  $x_1 = 3$ ,  $x_2 = 2$ .
```

Вывод.

В ходе проделанной работы, я изучил и приобрел навыки работы с методами класса.

Code review. (by zzzcode.ai)

Резюме

Данный код реализует несколько математических операций, включая вычисление площади треугольника, решение квадратного уравнения и вычисление факториала. Код написан на C# и включает в себя классы `Operation`, `Program` и `Utils`. В целом, код выполняет свои функции, но требует улучшений в области обработки ошибок, читаемости и структуры.

Ошибка

В методе `TriangleArea` возвращается `-1` в случае, если треугольник не существует. Это может вызвать путаницу, так как `-1` может быть

интерпретировано как валидное значение площади. Рекомендуется использовать исключения для обработки ошибок.

Стиль кода

Код в целом следует стандартам C#, однако есть некоторые моменты, которые можно улучшить:

Использование `var` для переменных, где тип очевиден. Избегание магических чисел, таких как `-1`, для обозначения ошибок.

Структура кода

Классы и методы организованы логически, однако стоит рассмотреть возможность разделения функциональности на более мелкие классы или модули для улучшения поддержки и тестирования.

Читаемость

Код читаем, но можно улучшить его с помощью:

Добавления комментариев для сложных участков кода. Использования более описательных имен переменных и методов.

Производительность

Код работает эффективно для небольших входных данных. Однако, для больших значений, особенно в методе `Factorial`, стоит рассмотреть использование более эффективных алгоритмов или библиотек.

Масштабируемость

Код может быть масштабирован, но для этого потребуется рефакторинг. Например, можно использовать интерфейсы и абстракции для улучшения гибкости.

Безопасность

Код не содержит явных уязвимостей, но стоит добавить проверки на вводимые данные, чтобы избежать исключений при некорректном вводе.

Обработка ошибок

Обработка ошибок в коде минимальна. Рекомендуется использовать исключения для более четкой обработки ошибок, а также добавлять проверки на вводимые данные.

Заключение

Код выполняет свои функции, но требует улучшений в области обработки ошибок, читаемости и структуры. Рекомендуется провести рефакторинг для повышения качества кода и его поддержки в будущем.