

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

Факультет инфокоммуникационных технологий

Направление подготовки 11.03.02

Лабораторная работа №1

Создание программы с помощью среды разработки VisualStudio.NET

Выполнил:

Доценников Никита Андреевич

Группа: К3221

Проверил:

Иванов Сергей Евгеньевич

Санкт-Петербург

2025

Цель работы:

- Ознакомиться с основами языка программирования C# и .NET.
- Научиться создавать и запускать консольные приложения.
- Получить навыки отладки программ.
- Освоить базовые операции ввода/вывода и обработку исключений.
- Реализовать простые задачи в виде консольных программ.

Задачи:

- Реализовать программу «Hello, Name», которая считывает имя пользователя и выводит приветствие.
- Создать проект в среде разработки (или с помощью .NET CLI) и реализовать аналогичную программу Greeter.
- Провести отладку программы с использованием точек останова и просмотра переменных.
- Написать программу Divider, которая выполняет деление целых чисел с обработкой исключений.
- Реализовать программу вычисления площади равностороннего треугольника по заданному периметру.

Теория:

- Язык C# является объектно-ориентированным языком, работающим в рамках платформы .NET.
- Консольные приложения содержат метод Main(), являющийся точкой входа программы.
- Для работы с консолью используются методы Console.WriteLine() и Console.ReadLine().
- Обработка ошибок выполняется с помощью конструкции try-catch.
- Для вычисления площади равностороннего треугольника можно использовать формулу Герона:

$$S = \sqrt{p(p-a)(p-b)(p-c)}, \quad p = \frac{P}{2}$$

где P - периметр треугольника.

Практическая часть.

Hello, Name

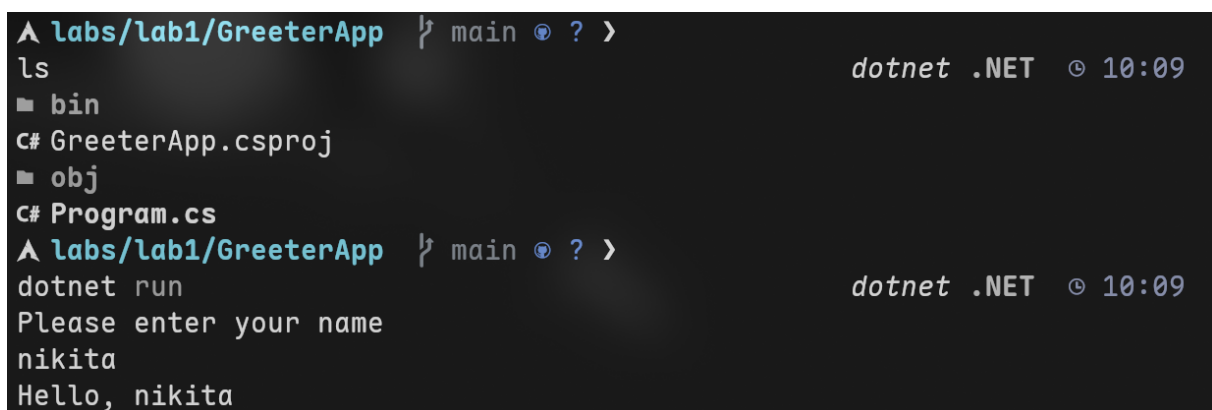
```
using System;
```

```

class Program
{
    static void Main()
    {
        string myName;
        Console.WriteLine("enter your name");
        myName = Console.ReadLine();
        Console.WriteLine("hello, {0}", myName);
    }
}

```

Пример работы:



```

^ labs/lab1/GreeterApp  ↵ main ? >
ls                                                                dotnet .NET  10:09
└─ bin
└─ obj
└─ GreeterApp.csproj
└─ Program.cs
^ labs/lab1/GreeterApp  ↵ main ? >
dotnet run                                                        dotnet .NET  10:09
Please enter your name
nikita
Hello, nikita

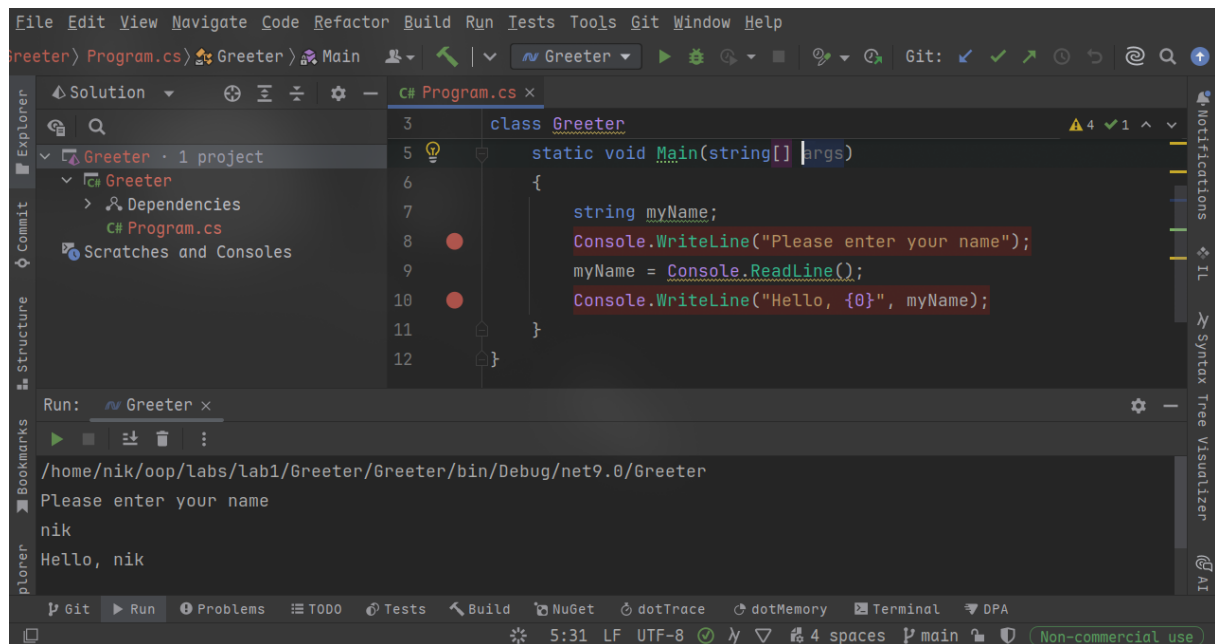
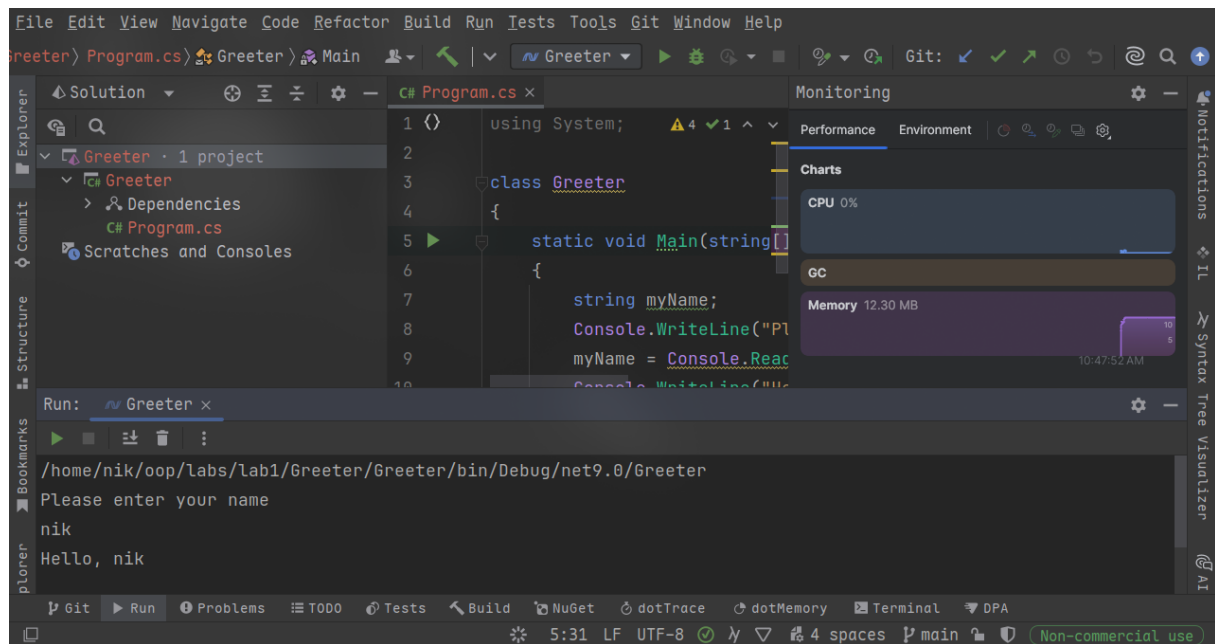
```

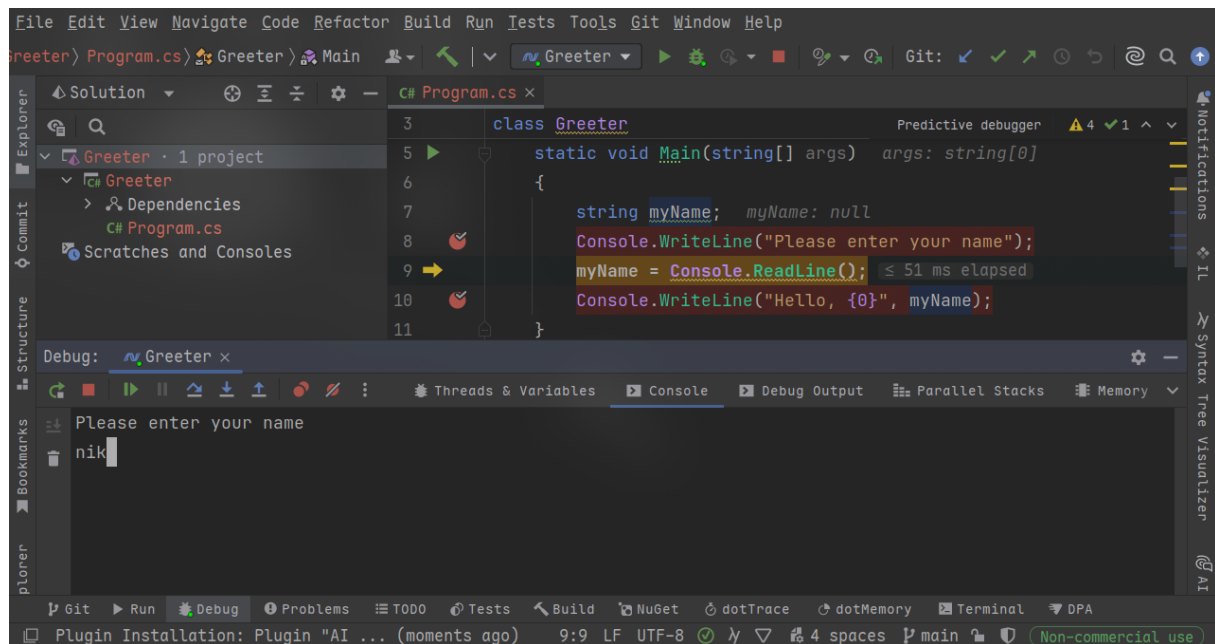
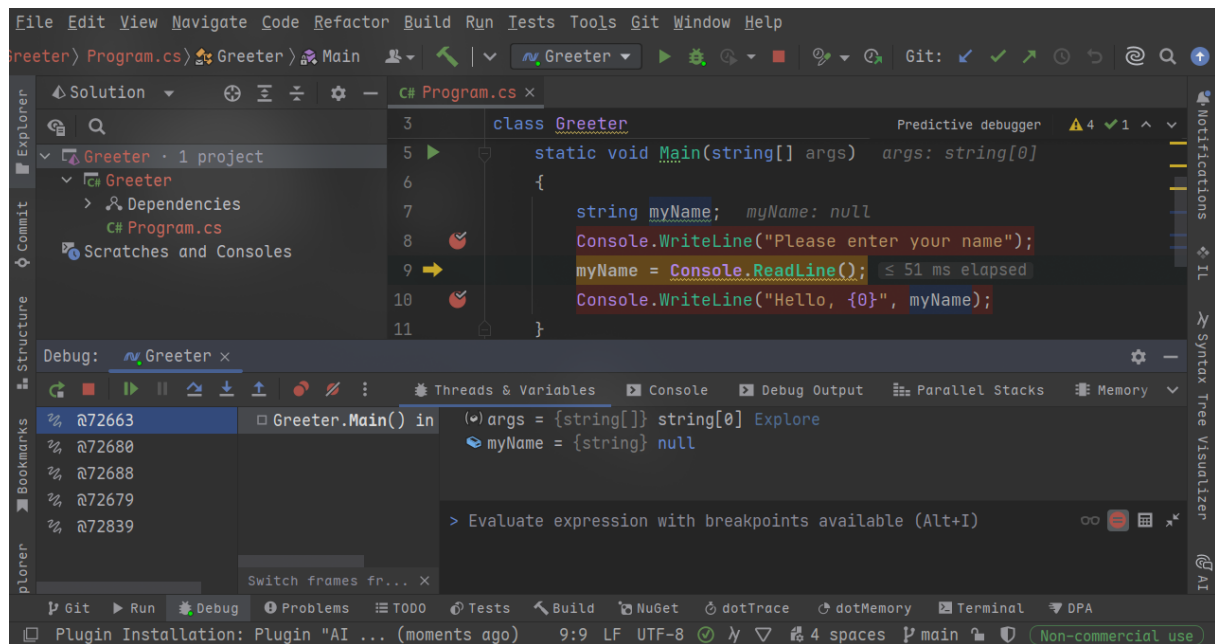
Debug:

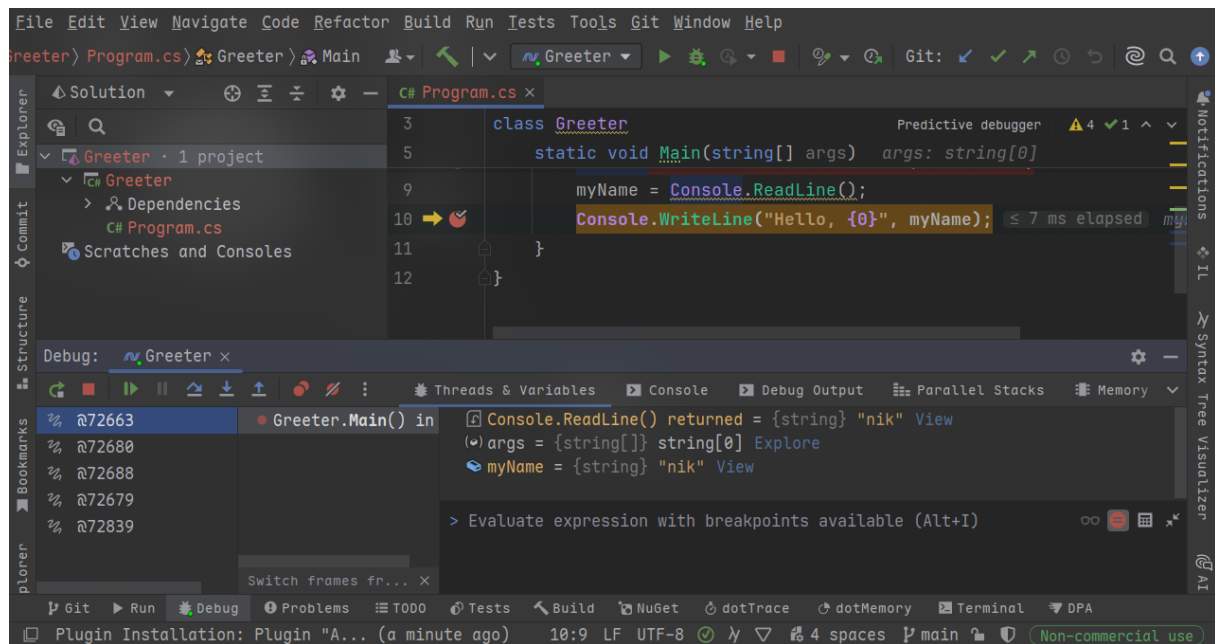
Ход выполнения:

- В JetBrains Rider был создан новый проект типа Console Application.
- В файл Program.cs добавлен код программы.
- Программа была запущена через встроенный терминал IDE.
- Для проверки корректности работы была введена строка с именем. Программа вывела корректное приветствие.
- Для изучения процесса отладки были установлены точки останова (breakpoints) на строках ввода и вывода.
- С помощью пошагового выполнения (Step Over/Step Into) наблюдалось изменение значения переменной myName.
- В окне Debug и панели Variables было подтверждено, что введённое значение корректно сохраняется в переменной.

Пример:







Divider

using System;

class DivideIt

```

{
    public static void Main(string[] args)
    {
        try
        {
            Console.WriteLine("enter the first integer");
            string temp = Console.ReadLine();
            int i = Int32.Parse(temp);
            Console.WriteLine("enter the second integer");
            temp = Console.ReadLine();
            int j = Int32.Parse(temp);
            int k = i / j;
            Console.WriteLine("{0} / {1} = {2}", i, j, k);
        }
        catch (FormatException e)
        {
            Console.WriteLine("format exception was thrown: {0}",
e.Message);
        }
        catch (Exception e)
        {
            Console.WriteLine("exception was thrown: {0}",
e.Message);
        }
    }
}

```

```

    }
}
}

```

Пример:

```

^ labs/lab1/DividerApp |> main @ ? >
dotnet run                                     dotnet .NET @ 10:13
enter the first integer
10
enter the second integer
5
10 / 5 = 2

```

```

^ labs/lab1/DividerApp |> main @ ? >
dotnet run                                     dotnet .NET @ 10:13
enter the first integer
10
enter the second integer
0
exception was thrown: Attempted to divide by zero.

```

```

^ labs/lab1/DividerApp |> main @ ? >
dotnet run                                     dotnet .NET @ 10:13
enter the first integer
abra
format exception was thrown: The input string 'abra' was not in a correct format.

```

TriangleArea

```

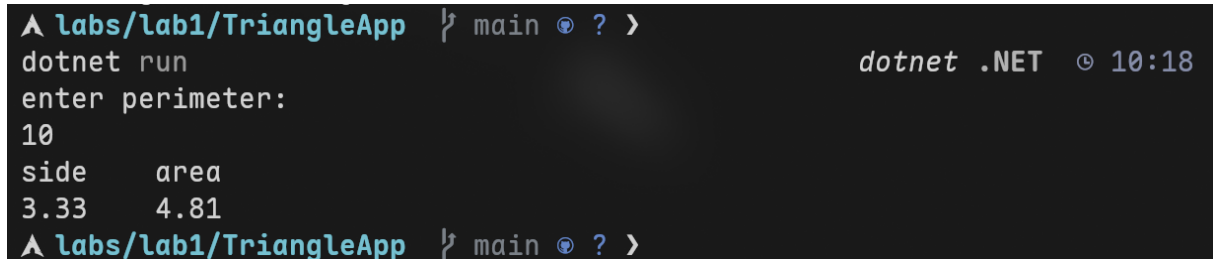
using System;
using System.Globalization;

class TriangleArea
{
    static void Main()
    {
        Console.WriteLine("perimeter");
        double p = double.Parse(Console.ReadLine(),
CultureInfo.InvariantCulture);
        double a = p / 3.0;
        double s = Math.Sqrt((1.5 * a) * (0.5 * a) * (0.5 * a) *
(0.5 * a));
        Console.WriteLine("side \tarea");
        Console.WriteLine("{0}\t{1}", a.ToString("F2",
CultureInfo.InvariantCulture), s.ToString("F2",

```

```
CultureInfo.InvariantCulture));  
    }  
}
```

Пример:



```
^ labs/lab1/TriangleApp | main @ ? >  
dotnet run  
enter perimeter:  
10  
side    area  
3.33    4.81  
^ labs/lab1/TriangleApp | main @ ? >
```

Выводы:

В ходе выполнения лабораторной работы:

- Освоены основы работы с консольными приложениями на C#.
- Получены навыки ввода/вывода данных, обработки исключений.
- Реализованы несколько программ, решающие простые задачи.
- Освоены базовые приёмы отладки приложений.

Code review:

Резюме

В данном коде представлены три класса: `DivideIt`, `Greeter` и `TriangleArea`. Каждый из них выполняет свою задачу, однако в коде есть несколько моментов, которые можно улучшить для повышения его качества и надежности.

Ошибка

В классе `DivideIt` отсутствует обработка деления на ноль, что может привести к исключению `DivideByZeroException`. Рекомендуется добавить проверку перед выполнением деления.

Стиль кода

Код в целом написан в соответствии с общепринятыми стандартами C#. Однако, для улучшения читаемости, стоит использовать более понятные сообщения для пользователя. Например, вместо “enter the first integer” можно использовать “Введите первое целое число”.

Структура кода

Каждый класс имеет свою собственную функцию Main, что является хорошей практикой. Однако, стоит рассмотреть возможность вынесения логики в отдельные методы для повышения модульности и повторного использования кода.

Читаемость

Читаемость кода можно улучшить, добавив комментарии, объясняющие логику работы программы. Это поможет другим разработчикам быстрее понять, что делает код.

Производительность

Код не содержит явных проблем с производительностью, однако, использование Int32.Parse и double.Parse без предварительной проверки входных данных может привести к исключениям, что негативно скажется на производительности в случае частых ошибок ввода.

Масштабируемость

Код не очень масштабируемый, так как каждая программа выполняет только одну задачу. Для более сложных приложений стоит рассмотреть использование классов и методов для разделения логики.

Безопасность

Обработка исключений в коде присутствует, но стоит добавить более специфическую обработку для DivideByZeroException в классе DivideIt. Это поможет избежать потенциальных проблем при делении на ноль.

Обработка ошибок

Обработка ошибок реализована, но стоит улучшить сообщения об ошибках, чтобы они были более информативными. Например, можно указать, что именно было введено неверно.

Заключение

В целом, код выполняет свои функции, но требует доработки для повышения надежности, читаемости и масштабируемости. Рекомендуется внести предложенные изменения, чтобы улучшить качество кода и сделать его более устойчивым к ошибкам.