МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
Кафедра интеллектуальных информационных технологий

Отчет по лабораторной работе №3

По дисциплине «Современные платформы программирования»
Специальность ПО-8

Выполнил:
Соколовский Н.И.
студент группы ПО-8
Проверил:
ст. преп. кафедры ИИТ,
«__»_____2024 г.

Брест 2024

Цель работы: научиться создавать и использовать классы в программах на языке программирования C#

Вариант 21

Задание 1.

Равнобедренный треугольник, заданный длинами сторон – Предусмотреть возможность определения площади и периметра, а так же логический метод, определяющий существует или такой треугольник. Конструктор должен позволить создавать объекты с начальной инициализацией. Реализовать метод equals, выполняющий сравнение объектов данного типа.

Код программы

```
using lab3z1;

var t1 = new Triangle();
var t2 = new Triangle(5,5,2);
Console.WriteLine(t2);
Console.WriteLine(t2.Exists());
Console.WriteLine(t2.Equals(t1));
Console.WriteLine(t2.CalcP());
Console.WriteLine(t2.CalcS());
```

Пример

```
A:5, B:5, C:2
True
False
12
4,898979485566356
```

Задание 2. Автоматизированная система в библиотеке

Написать стековый калькулятор, который принимает в качестве аргумента командой строки
имя файла, содержащего команды. Если аргумента нет, то использовать стандартный поток
ввода для чтения команд. Для вычислений допускается использовать вещественные числа.

Реализовать следующий набор команд:

• # – строка с комментарием.

• POP , PUSH – снять/положить число со/на стек(а).

• + , - , * , /, SQRT – арифметические операции. Используют один или два верхних элемента
стека, изымают их из стека, помещая результат назад

• PRINT – печать верхнего элемента стека (без удаления).

• DEFINE – задать значение параметра. В дальнейшем везде использовать вместо параметра это значение.

Содержимое стека и список определенных именованных параметров передавать команде в виде специального объекта – контекста исполнения. Разработать группу классов исключений, которые будут выбрасывать команды при исполнении. В случае возникновения исключения – выводить информацию об ошибке и продолжать исполнение программы (из файла или команд вводимых с консоли)

Код программы

```
using System;
using System.Collections.Generic;

namespace StackCalculator
{
    class Program
    {
        static void Main(string[] args)
        {

            string input = "";
            if (args.Length > 0)
            {
                // Чтение команд из файла
                input = File.ReadAllText(args[0]);
            }
            else
            {
                // Чтение команд из стандартного ввода
                Console.WriteLine("Enter commands... {Type 'exit' to end}");
                while (true)
                {
                    string str = Console.ReadLine();
                    if (str != "exit")
                    {
                        input += Console.ReadLine();
                        input += '\n';
                    }

                    break;

                }
            }
            string[] commands = input.Split('\n',
StringSplitOptions.RemoveEmptyEntries);
            foreach (string command in commands)
            {
                string[] commandText = command.Split(' ');
                switch (commandText[0].Trim().ToUpper())
                {
                    case "#":
                        // Комментарий
                        break;
                    case "POP":
                        Context.Stack.Pop();
                        break;
                    case "PUSH":
                        if (commandText.Length != 2)
                        {
                            MyException.ErrorWrongCountOfArgument(command);
                        }
                        else
                        {
                            double value;
```

```
                            if (double.TryParse(commandText[1], out value) ||
Context.Parameters.TryGetValue(commandText[1], out value))
                            {
                                Context.Stack.Push(value);
                            }
                            else
                            {

MyException.ErrorVariableNotFound(commandText[1]);
                            }

                        }

                        break;
                    case "+":
                        switch (commandText.Length)
                        {
                            case 1:
                                Context.Stack.Push(Context.Stack.Pop() +
Context.Stack.Pop());
                                break;
                            case 2:
                                double value;
                                if (double.TryParse(commandText[1], out
value) || Context.Parameters.TryGetValue(commandText[1], out value))
                                {
                                    Context.Stack.Push(Context.Stack.Pop() +
value);
                                }
                                else
                                {

MyException.ErrorVariableNotFound(commandText[1]);
                                }
                                break;
                            case 3:
                                double value1 , value2;
                                if ((double.TryParse(commandText[1], out
value1) && double.TryParse(commandText[2], out value2)) ||
(Context.Parameters.TryGetValue(commandText[1], out value1) &&
Context.Parameters.TryGetValue(commandText[1], out value2)))
                                {
                                    Context.Stack.Push(value1 + value2);
                                }
                                else
                                {

MyException.ErrorVariableNotFound(command);
                                }
                                break;
                            default:

MyException.ErrorWrongCountOfArgument(command);
                                break;
                        }
                        break;
                    case "-":
                        switch (commandText.Length)
                        {
                            case 1:
                                Context.Stack.Push(Context.Stack.Pop() -
Context.Stack.Pop());
                                break;
                            case 2:
```

```
                                double value;
                                if (double.TryParse(commandText[1], out
value) || Context.Parameters.TryGetValue(commandText[1], out value))
                                {
                                    Context.Stack.Push(Context.Stack.Pop() -
value);
                                }
                                else
                                {

MyException.ErrorVariableNotFound(commandText[1]);
                                }
                                break;
                            case 3:
                                double value1 , value2;
                                if ((double.TryParse(commandText[1], out
value1) && double.TryParse(commandText[2], out value2)) ||
(Context.Parameters.TryGetValue(commandText[1], out value1) &&
Context.Parameters.TryGetValue(commandText[1], out value2)))
                                {
                                    Context.Stack.Push(value1 - value2);
                                }
                                else
                                {

MyException.ErrorVariableNotFound(command);
                                }
                                break;
                            default:

MyException.ErrorWrongCountOfArgument(command);
                                break;
                        }
                        break;

                    case "*":
                        switch (commandText.Length)
                        {
                            case 1:
                                Context.Stack.Push(Context.Stack.Pop() *
Context.Stack.Pop());
                                break;
                            case 2:
                                double value;
                                if (double.TryParse(commandText[1], out
value) || Context.Parameters.TryGetValue(commandText[1], out value))
                                {
                                    Context.Stack.Push(Context.Stack.Pop() *
value);
                                }
                                else
                                {

MyException.ErrorVariableNotFound(commandText[1]);
                                }
                                break;
                            case 3:
                                double value1 , value2;
                                if ((double.TryParse(commandText[1], out
value1) && double.TryParse(commandText[2], out value2)) ||
(Context.Parameters.TryGetValue(commandText[1], out value1) &&
Context.Parameters.TryGetValue(commandText[1], out value2)))
                                {
                                    Context.Stack.Push(value1 * value2);
```

```
                                }
                                else
                                {

MyException.ErrorVariableNotFound(command);
                                }
                                break;
                            default:

MyException.ErrorWrongCountOfArgument(command);
                                break;
                        }
                        break;
                    case "/":
                        switch (commandText.Length)
                        {
                            case 1:
                                Context.Stack.Push(Context.Stack.Pop() /
Context.Stack.Pop());
                                break;
                            case 2:
                                double value;
                                if (double.TryParse(commandText[1], out
value) || Context.Parameters.TryGetValue(commandText[1], out value))
                                {
                                    Context.Stack.Push(Context.Stack.Pop() /
value);
                                }
                                else
                                {

MyException.ErrorVariableNotFound(commandText[1]);
                                }
                                break;
                            case 3:
                                double value1 , value2;
                                if ((double.TryParse(commandText[1], out
value1) && double.TryParse(commandText[2], out value2)) ||
(Context.Parameters.TryGetValue(commandText[1], out value1) &&
Context.Parameters.TryGetValue(commandText[1], out value2)))
                                {
                                    Context.Stack.Push(value1 / value2);
                                }
                                else
                                {

MyException.ErrorVariableNotFound(command);
                                }
                                break;
                            default:

MyException.ErrorWrongCountOfArgument(command);
                                break;
                        }
                        break;
                    case "SQRT":
                        if (commandText.Length != 2)
                        {
                            MyException.ErrorWrongCountOfArgument(command);
                        }
                        else
                        {
                            double value;
                            if (double.TryParse(commandText[1], out value) ||
```

```
Context.Parameters.TryGetValue(commandText[1], out value))
                            {
                                Context.Stack.Push(Math.Sqrt(value));
                            }
                            else
                            {

MyException.ErrorVariableNotFound(commandText[1]);
                            }

                        }
                        break;
                    case "PRINT":
                        Console.WriteLine($"Element on top:
{Context.Stack.Peek()}");
                        break;
                    case "DEFINE":
                        if (commandText.Length != 2)
                        {
                            MyException.ErrorWrongCountOfArgument(command);
                        }
                        else
                        {
                            string paramName = commandText[1];
                            double paramValue = Context.Stack.Pop();
                            Context.Parameters[paramName] = paramValue;
                        }
                        break;
                    default:
                        MyException.ErrorCommandNotFound(command);
                        break;
                }
            }


        }
    }
}
```

Пример

```
PUSH 10
PUSH 2
+
PUSH 40
DEFINE KEK
PUSH 30
+ KEK
PRINT
POP
PRINT
```

```
Element on top: 70
Element on top: 12
```

Вывод

Научились создавать и использовать классы в программах на языке программирования C#.