**Лабораторная работа 2. Использование делегатов**

В данной лабораторной работе нужно создать класс калькулятор, независимо от того, какой интерфейс будет у приложения. Приложение может быть консольным, и в нем вводится одно выражение для вычисления. Приложение может быть стандартным оконным, где числа и знаки вводятся с помощью кнопок или клавиш мыши.

Создайте проект библиотеки классов с именем CalcLibrary. Добавьте в него статический класс Calc.

Метод, который будет проводить вычисления DoOperation, принимает строку с выражением и возвращает строку с ответом:

public static string DoOperation(string s)

{

//получить операнды

//получить операцию

return //вычислить и получить строку;

}

Этот метод нужно сделать публичным, остальные методы будут доступны только внутри проекта.

В методе получения массива операндов достаточно выделить из исходной строки подстроки из чисел.

static string[] GetOperands(string s)

Будем использовать регулярные выражения. Для начала подключите необходимое пространство имен:

using System.Text.RegularExpressions;

Затем в теле метода составьте шаблон, по которому будете искать:

string pattern = @"[^\d,\.]";

По данному шаблону будут найдены все символы, кроме: символов цифр, точки и запятой. Подстрока "\d" означает набор цифр, чтобы символ точки означал именно символ точки, как знака, его необходимо экранировать. Квадратные скобки означают набор символов, то есть любой символ из набора. Знак ^ означает отрицание, то есть исключает любой символ из набора. Но по этому шаблону невозможно работать с отрицательными числами. Исправьте этот недостаток.

Далее по данному шаблону нужно создать регулярное выражение – экземпляр класса Regex:

Regex rgx = new Regex(@"[^\d,\.]");

Получить коллекцию всех подходящих под шаблон выражений:

MatchCollection mc = rgx.Matches(s);

И преобразовать ее в список строк:

List<string> lm = new List<string>();

foreach (Match m in mc)

{

lm.Add(m.Value);

}

В полученном списке будут содержаться все знаки, кроме чисел и точки/запятой. То есть все разделительные знаки для получения нужных подстрок из чисел. Остается только разделить исходную строку на подстроки и вернуть полученный массив:

s.Split(lm.ToArray(), StringSplitOptions.None);

Для проверки правильности работы методов добавьте к решению тестовый проект. Проверьте правильность выполнения метода GetOperands.

[TestMethod]

public void CalculateTestMethod()

{

String[] a = Calc.GetOperands("23+4,5");

Assert.AreEqual("23", a[0]);

Assert.AreEqual("4,5", a[1]);

}

**Запуск**. Запустите тестирование метода GetOperands.

Метод получения операции из исходной строки напишите самостоятельно:

static string GetOperation(string s)

Добавьте тест для проверки корректности возвращаемого значения из метода GetOperation.

**Запуск**. Запустите тестирование метода GetOperation.

Добавьте соответствующие вызовы методов в метод DoOperation:

string[] operands = GetOperands(s);

string op = GetOperation(s);

Остается выполнить операцию и получить результат. Реализуем выполнение бинарных операций: +, –, \*, /.

Для реализации операций можно было бы использовать оператор выбора:

witch (op)

{

case "+": return x + y;

case "-": return x - y;

case "\*": return x\* y;

case "/": return x / y;

}

Но вместо него будем использовать делегаты. Все операции принимают два операнда и возвращают одно значения. Типы совпадают.

В классе Calc опишите:

delegate T OperationDelegate<T>(T x, T y);

Добавьте в класс словарь, с помощью которого можно будет выбрать соответствующую операцию:

static Dictionary<string, OperationDelegate<double>> DoubleOperation =

new Dictionary<string, OperationDelegate<double>>

{

{ "+", delegate(double x, double y){ return x + y; } },

{ "-", delegate(double x, double y){ return x - y; } },

{ "\*", delegate(double x, double y){ return x \* y; } },

{ "/", delegate(double x, double y){ return x / y; } },

};

Таким образом, метод DoOperation должен будет вернуть:

DoubleOperation[op](double.Parse(operands[0]), double.Parse(operands[1])).ToString();

Добавьте тест для проверки корректности возвращаемого значения из метода DoOperation.

[TestMethod]

public void ResultTestMethod()

{

Assert.AreEqual("27,5", Calc.doubleOperation["+"](23, 4.5).ToString());

string result = Calc.DoOperation("23+4,5");

Assert.AreEqual("27,5", result);

}

**Запуск**. Запустите тестирование метода DoOperation.

Если все работает, переходите к созданию интерфейса.

Добавьте новый WPF-проект к решению. Для начала разработки интерфейса для калькулятора можно воспользоваться примером: <https://metanit.com/sharp/wpf/1.2.php>. Но это лишь начальный интерфейс. Сделайте его удобнее. Например, измените направление текста в TextBlock. Добавьте в TextBlock начальное значение – "0". Шрифт крупнее и т.д.

Описание поведения программы. При нажатии на кнопку с цифрой или знаком действия содержание кнопки добавляется к содержимому в TextBlock. При нажатии на кнопку очистки, должна происходить очистка TextBlock, то есть в TextBlock появится текст с "0". При нажатии на кнопку с равенством происходить вычисление выражения и отображение результата.

**Запуск**. Запустите проект и убедитесь в его работоспособности.

Задание.

1. Вместо делегата можно использовать анонимный метод (лямбду). Замените в словаре OperationDelegate делегаты на анонимные методы. Например, делегат для сложения можно заменить на: (x, y)=>x + y.
2. При нажатии на кнопку с цифрой в строке остается незначащий ноль. Сделайте так, чтобы незначащий ноль удалялся при вводе цифр.
3. Добавьте кнопку для удаления последней цифры набранного числа в строке.
4. Добавьте к калькулятору новые возможности:
   1. нахождение остатка и целой части для целых чисел;
   2. смена знака (+/–);
   3. нахождение значения 1/х;
   4. нахождение факториала числа;
5. Воспользовавшись свойствами и методами класса Math, добавьте к калькулятору новые возможности:
   1. нахождение квадратного корня заданного числа;
   2. возведение числа в квадрат;
   3. нахождение введенной степени заданного числа;
   4. нахождение синуса, косинуса, тангенса заданного угла;
   5. вывод значения числа e;
   6. нахождение значения ex;
   7. вывод значения числа π.
6. Протестируйте приложение, выявите ошибки в работе программы, модифицируйте программу таким образом, чтобы неправильные действия пользователя не приводили к возникновению ошибок.