МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования  
«Тверской государственный технический университет» (ТвГТУ)

Кафедра “Программного обеспечения”

ЛАБОРАТОРНАЯ РАБОТА №1.

РАЗРАБОТКА ИНТЕРФЕЙСНЫХ ПРОЕКТОВ WPF

по дисциплине «Основы программирования»

Разработали: студенты группы

Б.ПИН.РИС.24.06

Думчев Данила Сергеевич

Еделькин Максим Викторович

Ткаченко Антон Александрович

Маматов Михаил Георгиевич

Ратхоре Александр Шарванович

Балабуев Никита Сергеевич

Проверил: Веревка Александр Андреевич

Тверь, 2025

**Оглавление**

[**Постановка задачи** 3](#_Toc210245362)

[**Архитектура приложения** 3](#_Toc210245363)

[**Трудности при разработке** 4](#_Toc210245364)

[**Обработка ошибок** 6](#_Toc210245365)

[**Элементы интерфейса** 7](#_Toc210245366)

[**Вывод** 10](#_Toc210245367)

[**Список литературы** 10](#_Toc210245368)

# **Постановка задачи**

Необходимо разработать приложение «Калькулятор», соответствующее следующим требованиям:

1. Необходимо реализовать калькулятор, предусматривающий операции: сложения, вычитания, умножения, деления.
2. Приложение состоит из двух страниц: калькулятор и история операций.
3. Во вкладке «Калькулятор» ввод производится исключительно при помощи кнопок.
4. Распределение кнопок и поля вывода цифр с помощью контейнера.
5. Наличие обработки ошибок и понятный интерфейс.
6. Во вкладке «История» должны сохраняться все операции, произведённые в текущем сеансе использования приложения. Операции должны сохраняться в формате равенства.

# **Архитектура приложения**

Архитектура приложения построена с использованием слоистой модели, что позволяет разделить интерфейс и вычислительную логику. Такой подход обеспечивает удобство сопровождения, расширяемость и повторное использование кода. Пользователь работает с графическим интерфейсом (WPF), а все операции выполняются в библиотеке классов, отвечающей за обработку данных.

Приложение включает два основных слоя: слой представления (UI), который отвечает за взаимодействие с пользователем, и слой бизнес-логики (BusinessLogic), реализованный как отдельная библиотека классов.

Слой представления представлен окном WPF и связан с бизнес-логикой через вызовы метода класса библиотеки. В нем содержатся элементы интерфейса: кнопки для ввода чисел и операций, текстовое поле для отображения выражений и результатов.

Слой бизнес-логики вынесен в отдельный проект библиотеки классов и служит ядром приложения. Основной компонент здесь — класс Calculator, содержащий методы для выполнения базовых арифметических операций (сложение, вычитание, умножение, деление).

| **Компонент (Модуль)** | **Файл** | **Ответственность** |
| --- | --- | --- |
| Слой представления (UI) | MainWindow.xaml, MainWindow.xaml.cs | Отвечает за построение интерфейса (кнопки, поле вывода), обработку нажатий и синхронизацию состояния. Логика ограничена передачей данных в бизнес-слой и отображением ответа. |
| Основной вычислительный модуль (BusinessLogic) | Calculator.cs | Содержит методы для выполнения арифметических операций. Является ядром приложения и обрабатывает все входные данные, переданные из интерфейса. |

# **Трудности при разработке**

Основная сложность во время разработки была связана с определением **логики и ограничений ввода данных** в пользовательском интерфейсе. Требовалось обеспечить интуитивно понятный и "безошибочный" ввод, чтобы предотвратить создание некорректных математических выражений пользователем.

Конкретные проблемы и их решения:

1. **Многократное повторение оператора при последовательном нажатии (+, -, \*, /)**.

**Проблема:** При многократном нажатии на кнопку оператора в строке ввода могли появиться выражения вида: 10++5.

**Решение:** В коде MainWindow.xaml.cs была реализована логика, которая **заменяет последний введенный оператор** на новый, если последний символ в строке currentInput уже является оператором. Это достигается с помощью метода IsLastCharOperator() и логики в обработчике OperatorButton\_Click.

1. **Повторение запятой при многократном нажатии или ее ввод в некорректном месте.**

**Проблема:** Ввод одной и более запятой в числе (10,,5; 10,5,2), постановка запятой сразу после оператора(10+,).

**Решение:** В обработчике CommaButton\_Click введены следующие проверки, использующие вспомогательные методы:

* + IsLastCharOperator(): Запрещает ввод запятой сразу после оператора.
  + IsLastCharComma(): Запрещает дублирование запятой (многократное нажатие).
  + HasCommaInCurrentNumber(): Запрещает ввод второй запятой в текущем числе, что гарантирует корректный формат десятичного числа.

1. **Ограничение на разрядность числа.**

**Проблема:** Неконтролируемый ввод большого количества цифр, что может привести к переполнению экрана.

**Решение:** Введено ограничение MaxDigits = 12. Метод GetLastNumberLength() подсчитывает длину последнего введённого числа. В обработчике NumberButton\_Click проверяется, что ввод новой цифры (или 00) не превысит установленный лимит.

1. **Корректное расположение и число скобок**

**Проблема:** Некорректное использование скобок - закрывающие без открывающих, несбалансированные выражения, синтаксические ошибки.

**Решение:** В системе реализован контроль скобок: метод Parenthesis\_Click() отслеживает баланс скобок через счетчик \_unclosedParentheses, блокируя некорректные закрывающие скобки и автоматически добавляя оператор умножения перед открывающей скобкой, если предыдущий символ является цифрой или закрывающей скобкой. При вычислении выражения метод EqualsButton\_Click() автоматически дополняет недостающие закрывающие скобки, предварительно удаляя последний оператор при необходимости.

# **Обработка ошибок**

Обработка ошибок реализована в статическом классе Calculator в файле **Calculator.cs**. Вычисления выполняются с использованием метода DataTable().Compute(), который позволяет безопасно вычислять математические выражения, заданные в виде строки.

Обработка ошибок происходит на двух уровнях:

1. **Проверка корректности выражения в C# коде:**

**Проблема:** **Деление на ноль** (10/0).

**Обработка:** Результат вычисления приводится к типу double. Проверяется, является ли он double.IsInfinity или double.IsNaN (что может произойти при делении на ноль).

**Результат:** Если обнаружено деление на ноль, метод возвращает строку **"Ошибка: деление на 0"**.

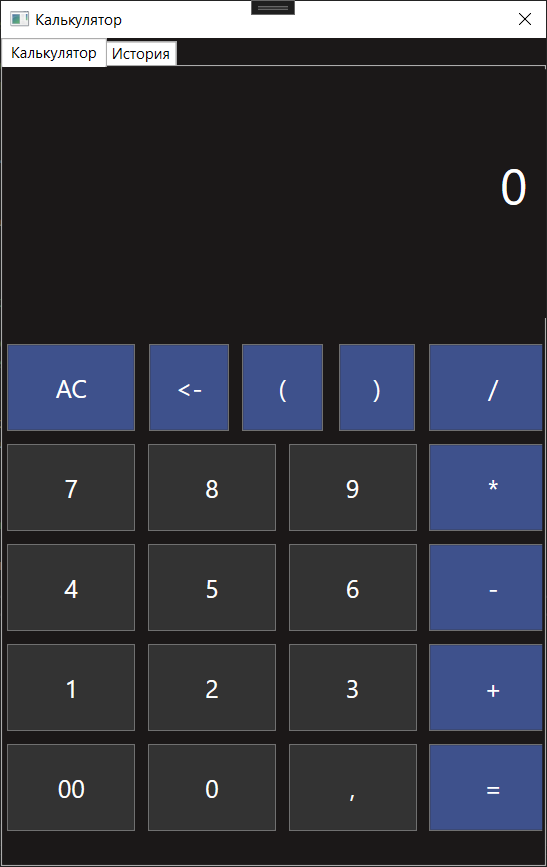
1. **Перехват исключений (Exception Handling) блоком try-catch:**

**Проблема:** **Переполнение** при работе с очень большими числами и непредвиденные ошибки вычисления.

**Обработка:** Перехватывается исключение OverflowException и общий catch блок.

**Результат:** Метод возвращает строку **"Ошибка: слишком большое число"** или строку **"Ошибка вычисления"** в остальных случаях

# **Элементы интерфейса**

Интерфейс приложения, описанный в файле **MainWindow.xaml**, построен с использованием стандартных элементов управления WPF. Ниже представлен внешний вид приложения и таблица с кратким описанием элементов интерфейса и примерами:

| Элемент интерфейса | Назначение и использование | Пример из XAML/CS |
| --- | --- | --- |
| **TabControl** | Контейнер, позволяющий разделить приложение на вкладки. В данном случае это **"Калькулятор"** и **"История"**. | <TabControl Background="#FF1B1818"> |
| **TabItem** | Непосредственно вкладка с заголовком (Header). | <TabItem Header="Калькулятор"> и <TabItem Header="История"> |
| **Grid** | Главный элемент для размещения других элементов, обеспечивает гибкую (строковую/столбцовую) компоновку кнопок и дисплея. | <Grid.RowDefinitions> и <Grid.ColumnDefinitions> для создания сетки кнопок. |
| **StackPanel** | Контейнер для размещения элементов друг за другом (в данном случае вертикально), использован для группировки дисплеев калькулятора. | <StackPanel VerticalAlignment="Top" Height="200"> |
| **TextBox** | Элемент для отображения текста. В приложении используется: |  |
| - DisplayTextBox | **Основное поле вывода**. Отображает текущий ввод или конечный результат. Использует FontSize="40", выравнивание по правому краю (TextAlignment="Right") и **только для чтения** (IsReadOnly="True"). | <TextBox x:Name="DisplayTextBox" Text="0" ... /> |
| - OperationTextBox | **Поле операции**. Отображает выражение после нажатия "=", чтобы пользователь видел, что было вычислено (пока оно в коде не используется, но зарезервировано для отображения выражения). | <TextBox x:Name="OperationTextBox" Text="" ... /> |
| - HistoryTextBox | **История вычислений** на отдельной вкладке. Имеет автоматический скроллинг (VerticalScrollBarVisibility="Auto"). Заполнение происходит в EqualsButton\_Click через HistoryTextBox.AppendText. | <TextBox x:Name="HistoryTextBox" ... /> |
| **Button** | Основной элемент для взаимодействия. Каждая кнопка (цифры, операторы, AC, =) имеет свой обработчик события Click. | <Button Content="7" Click="NumberButton\_Click" /> |

Ниже приведен пример полного кода для кнопки цифры калькулятора и взаимодействия с ней по нажатию в коде C# (MainWindow.xaml.cs) :

XAML:

<Button x:Name="seven"

Content="7"

Grid.Row="1"

Grid.Column="0"

Margin="5"

FontSize="20"

Background="#FF333333"

Foreground="White"

Click="NumberButton\_Click" />

С#:

private void NumberButton\_Click(object sender, RoutedEventArgs e)

{

var button = (Button)sender;

currentInput.Append(button.Content.ToString());

DisplayTextBox.Text = currentInput.ToString();

}

**Вывод**  
В рамках выполнения лабораторной работы №1 было успешно разработано WPF-приложение «Калькулятор» с разделением интерфейса и вычислительной логики.

В процессе работы были достигнуты цели:

1. Освоены и применены основные элементы WPF для построения удобного интерфейса калькулятора (Grid, Button, TextBox).
2. Реализована архитектура с разделением слоёв: интерфейс (MainWindow) и библиотека классов с бизнес-логикой (Calculator, AdvancedCalculator).
3. Обеспечена обработка ошибок и защита от некорректного пользовательского ввода (например, деление на ноль).

Созданный калькулятор выполняет базовые арифметические операции и может быть расширен дополнительными функциями за счёт модульной структуры бизнес-логики. Такое решение соответствует техническому заданию и демонстрирует преимущества использования WPF в сочетании с классической архитектурой разделения обязанностей.

# **Список литературы**

1. Больше информации о WPF <https://professorweb.ru/my/WPF/base_WPF/level1/base_WPF_index.php>
2. Конкретные примеры:  
   <https://github.com/Vik154/TUTORIAL_WPF>