

Лабораторная работа №2: Компоненты и события

Цель:

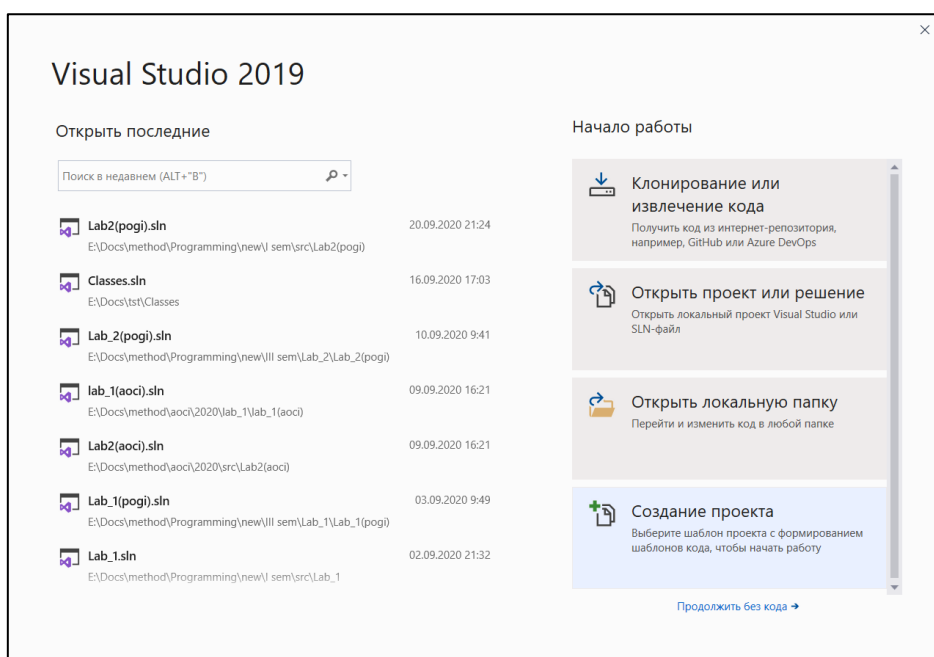
Целью данной работы является получение базовых навыков использования компонентов и описания событий на языке высокого уровня C# в среде программирования Microsoft Visual Studio 2017 Community

Справка:

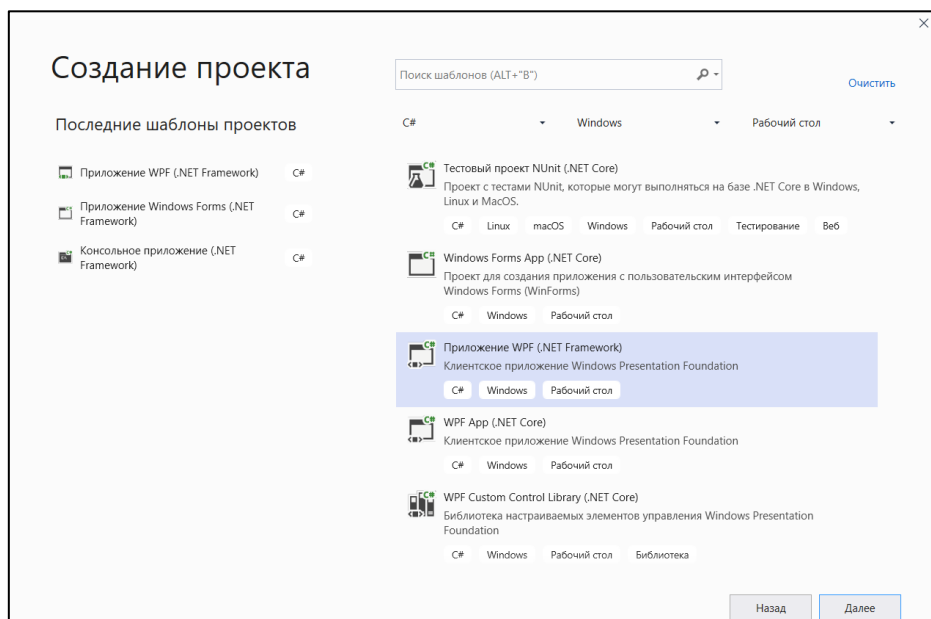
Среда разработки Microsoft Visual Studio 2019 Community доступна для скачивания на официальном сайте корпорации Microsoft по адресу: <https://www.visualstudio.com/ru/downloads/>

Создание приложения WPF в Microsoft Visual Studio 2019 Community:

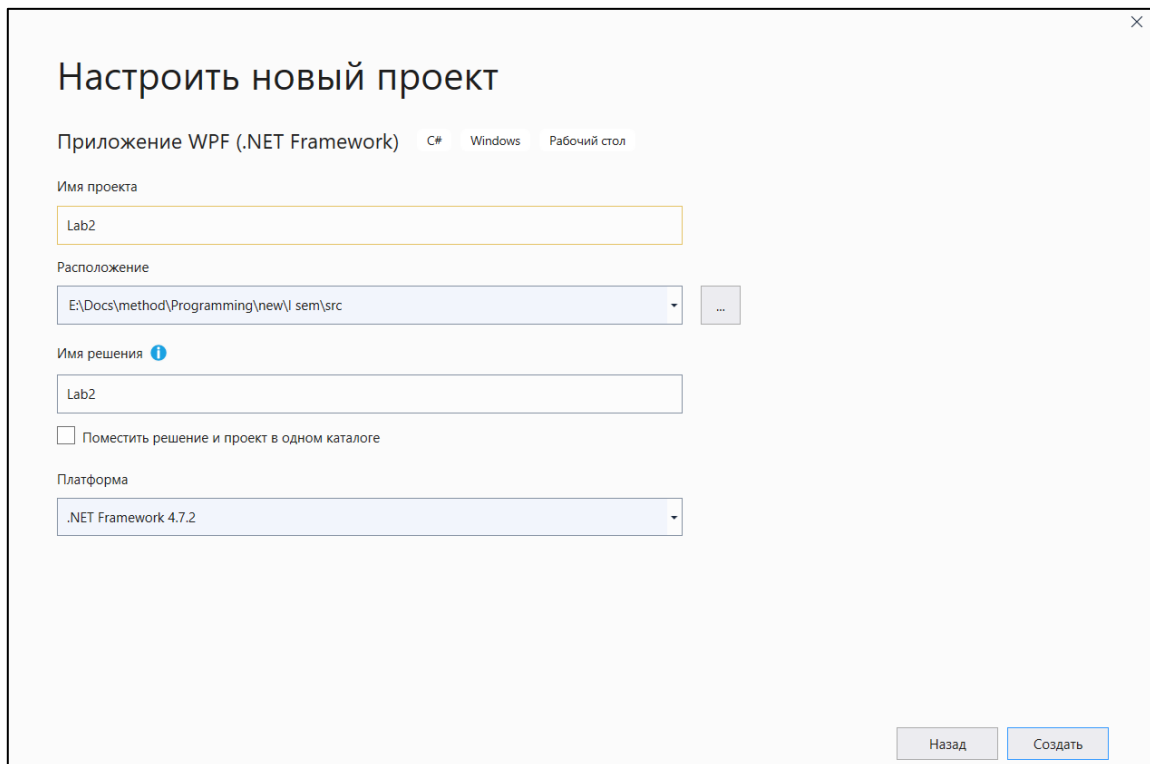
Запустите **Visual Studio** и выберите “Создание проекта”:



В появившемся окне, выберите тип проекта “Приложение WPF”:



Введите **название** проекта и его **расположение**:



Настроить новый проект

Приложение WPF (.NET Framework) C# Windows Рабочий стол

Имя проекта

Lab2

Расположение

E:\Docs\method\Programming\new\I sem\src

Имя решения

Lab2

☐ Поместить решение и проект в одном каталоге

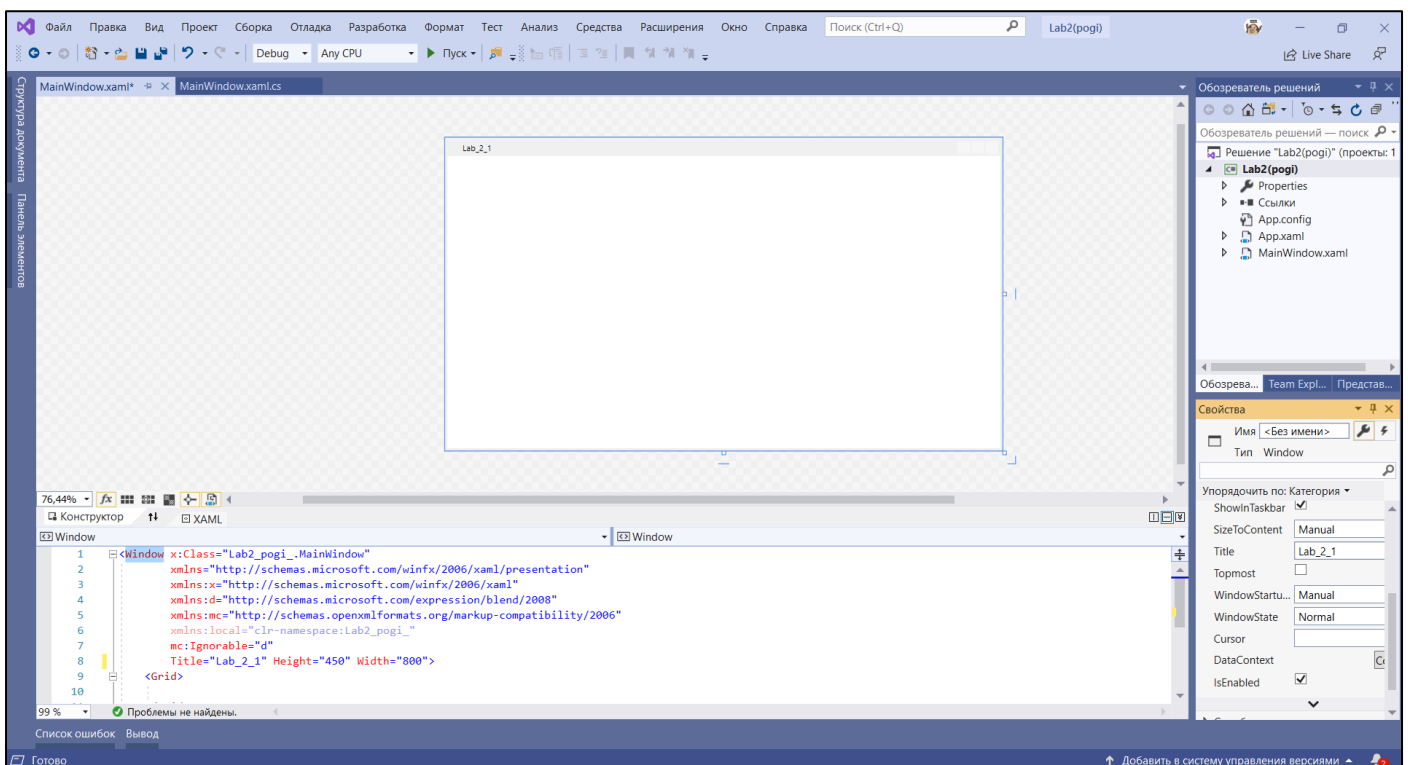
Платформа

.NET Framework 4.7.2

Назад Создать

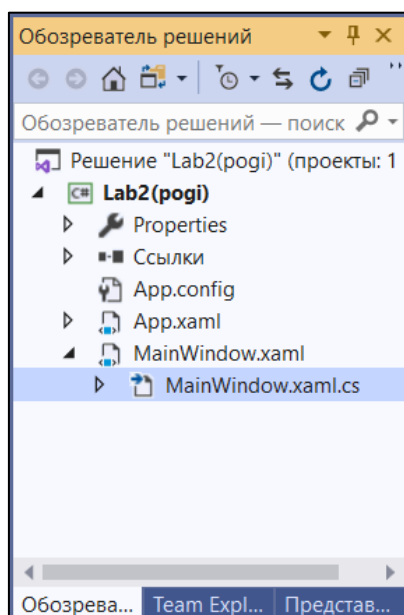
Примечание: Сохранять проект желательно не на системном диске (не на “C:\”) и не на переносном запоминающем устройстве (работа с “флешки” сильно замедляет процесс сборки и запуска приложения).

Если всё было сделано правильно, должен открыться визуальный редактор, а также файл с именем **MainWindow.xaml**:

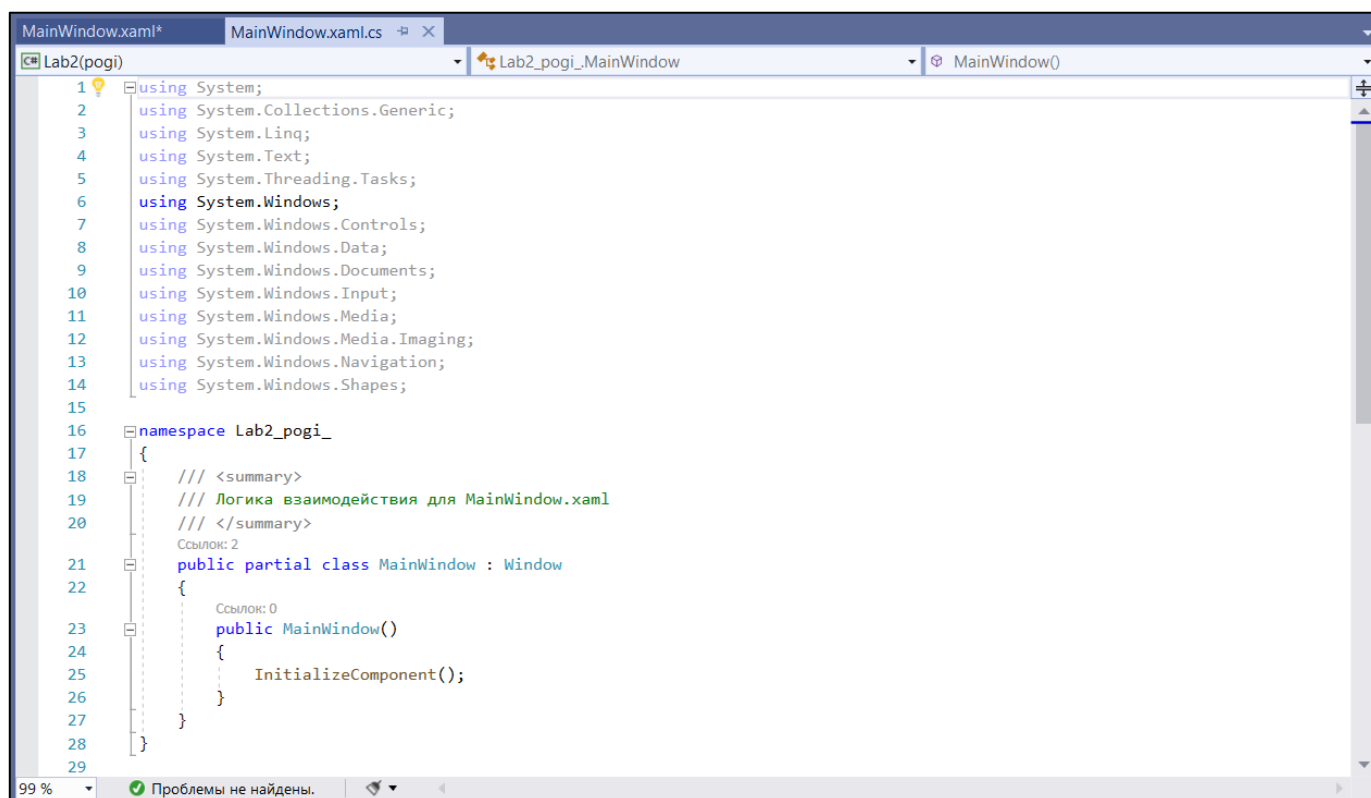


Файл **MainWindow.xaml** содержит описание визуального представления формы разрабатываемого приложения в формате **xaml**, а визуальный редактор позволяет увидеть графическое представление этого описания.

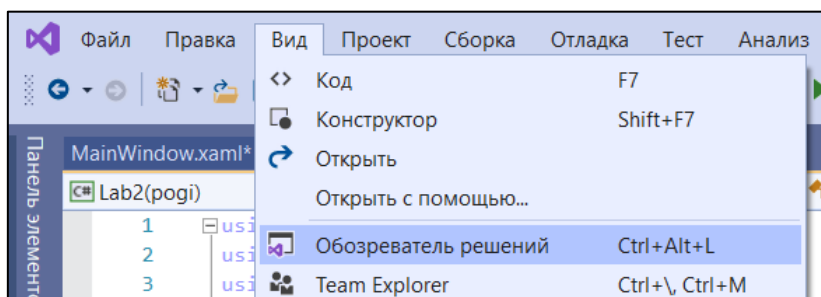
Переключаться между визуальным редактором и исходным кодом приложения можно, на пример, при помощи обозревателя решения:



Исходный код приложения содержится в файле **MainWindow.xaml.cs**.

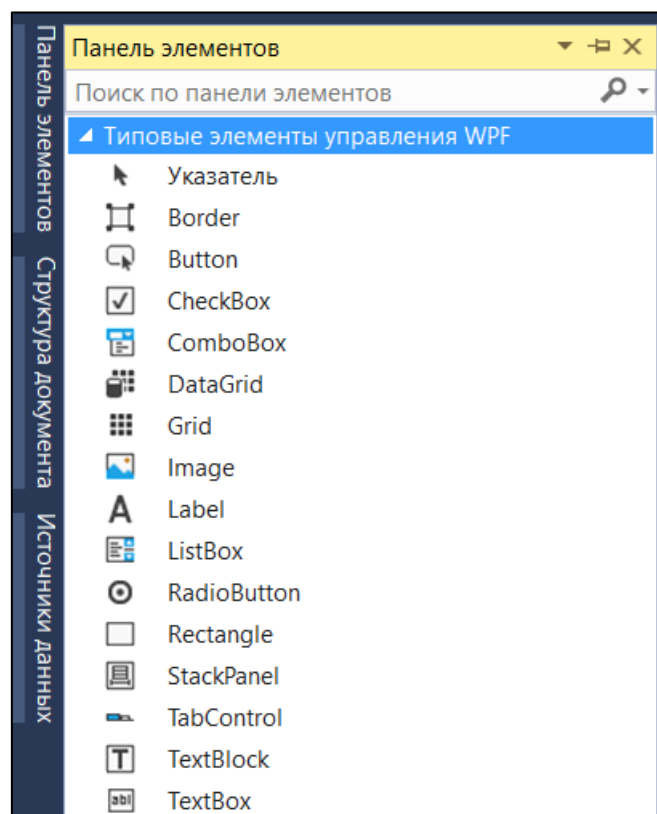


В случае если **Обозреватель решений** отсутствует на экране, добавить его можно выбрав **Вид -> Обозреватель решений**:



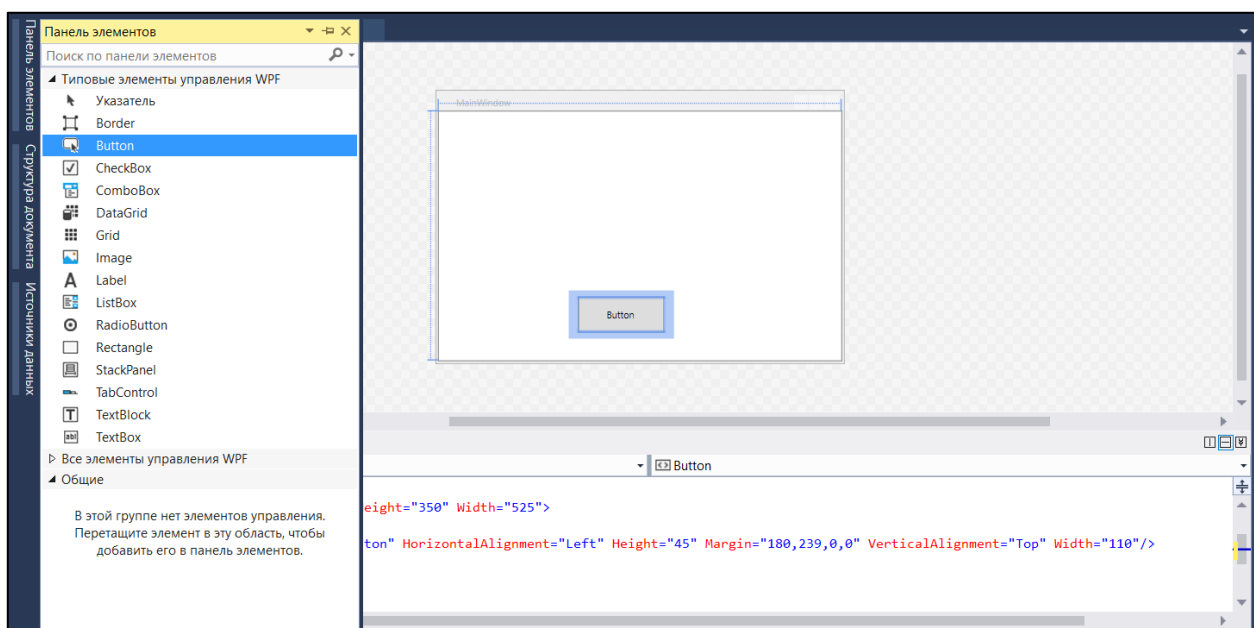
Добавление компонентов и запуск программы:

Для добавления компонентов на форму используется **Панель элементов**:

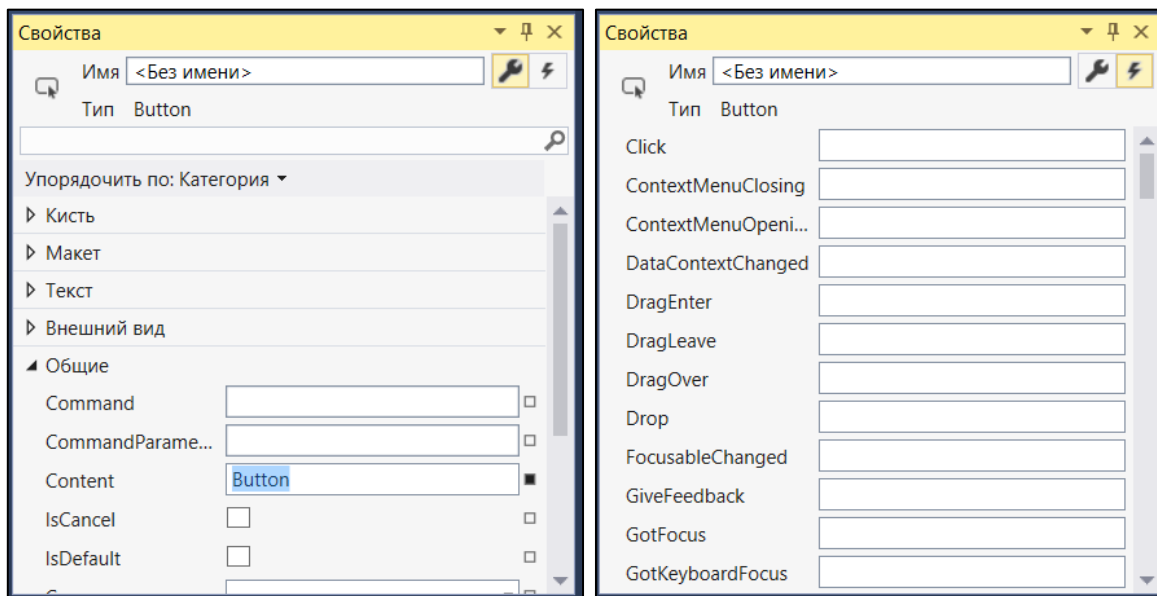


В случае если **Панель элементов** отсутствует на экране, добавить её можно выбрав **Вид -> Панель элементов**, либо **Вид -> Другие окна -> Панель элементов**

Для добавления компонента, выберите его на **Панели элементов** и разместите на форме в визуальном редакторе, используя манипулятор “мышь”:



При добавлении компонента на форму, в окне **Свойства**, можно посмотреть список доступных свойств и событий для этого компонента:



В случае если окно **Свойства** отсутствует на экране, добавить его можно выбрав **Вид -> Окно Свойств**, либо **Вид -> Другие окна -> Окно Свойств**.

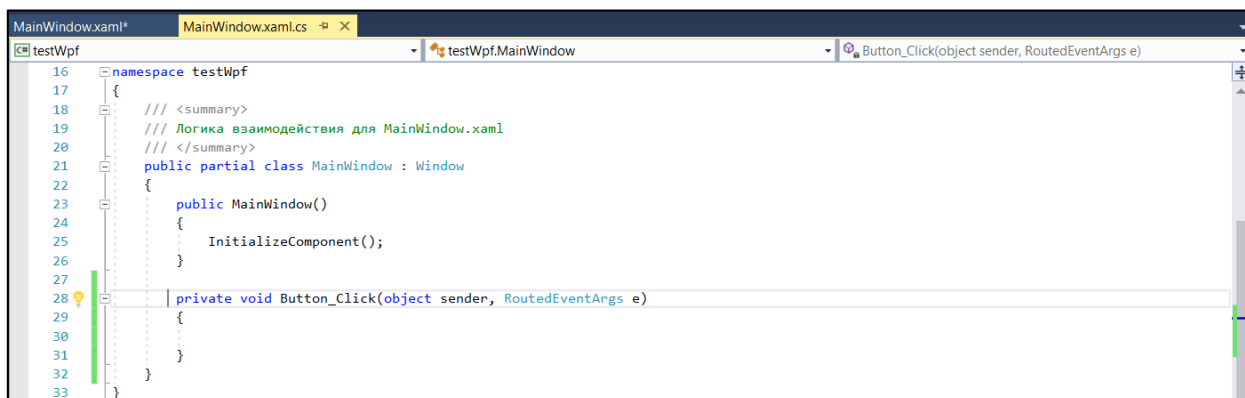
Вкладка свойств содержит визуальное и функциональное описание компонента (размер, позиция, цвет, ассоциация с внешними элементами и т.д.). Вкладка событий содержит список событий, генерируемых компонентом.

Например, компонент **Button** (кнопка), содержит следующие, часто используемые, свойства:

- Имя: имя присваиваемое компоненту.
- Content: текст надписи на кнопке.
- isEnabled: флаг, определяющий, является ли кнопка активной.

Наиболее часто используемым событием компонента **Button**, является **Click**, отвечающее за обработку события нажатия на кнопку (если она активна).

Добавить **обработчик события**, можно дважды кликнув напротив его названия в окне **Свойства**. Visual Studio автоматически генерирует код обработчика события в соответствующем файле:



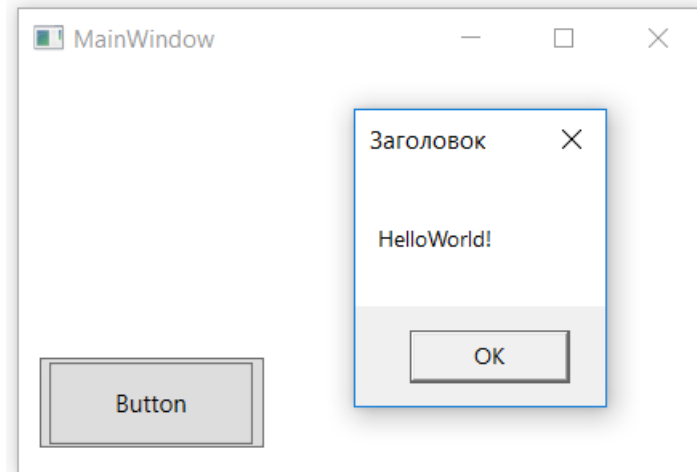
Обработчик события является методом, параметрами которого являются ссылка на вызвавший — это событие объект и список аргументов, сопутствующих событию (например, координаты курсора “мыши” если это обработчик события, связанного с “мышью”)

Для демонстрации работы программы, предлагается использовать класс **MessageBox**:

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    string st = "HelloWorld!";
    MessageBox.Show(st, "Заголовок", MessageBoxButton.OK);
}
```

Использованная перегрузка метода **Show**, класса **MessageBox**, выводит сообщение, переданное в качестве первого параметра, в окне сообщения с заголовком, переданным в качестве второго параметра и набором кнопок, указанным в качестве третьего параметра.

Результат работы приложения после нажатия кнопки:



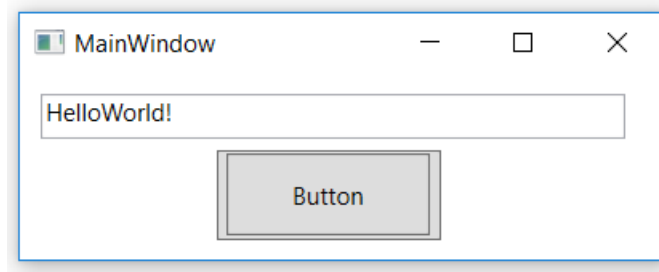
В случае, если в процессе запуска приложения появляется окно с отладочными функциями, выберите: **Перейти к визуальному дереву**, а затем отключите опцию **Показывать средства разработки в приложении**.

Обращение к компонентам осуществляется по их имени. Например, если добавить на форму компонент **TextBox** (поле для ввода текста), то можно получать доступ к содержащемуся в нём тексту при помощи свойства **Text**.

Пример обработчика события:

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    string st = "HelloWorld!";
    tb1.Text = st;
}
```

Результат, после нажатия кнопки:



Где **tb1** – имя компонента **TextBox**, заданное в окне свойств.

Следует помнить, что в свойстве **Text** лежат данные типа **string**.

Часто используемые компоненты:

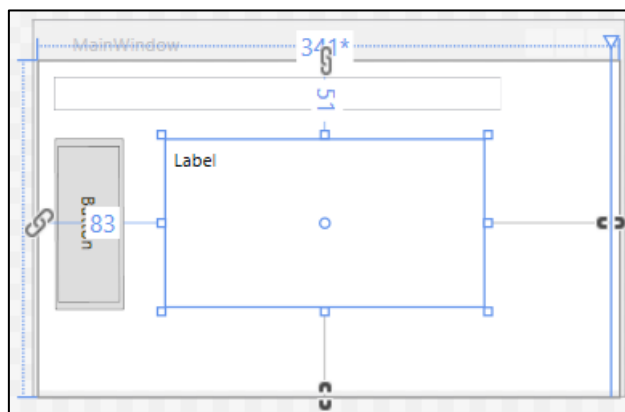
Далее будут описаны некоторые свойства и события часто используемых компонентов.

Label (поле для вывода текста)

Свойства:

Content – свойство, содержащее текст, отображаемый в области вывода текста.

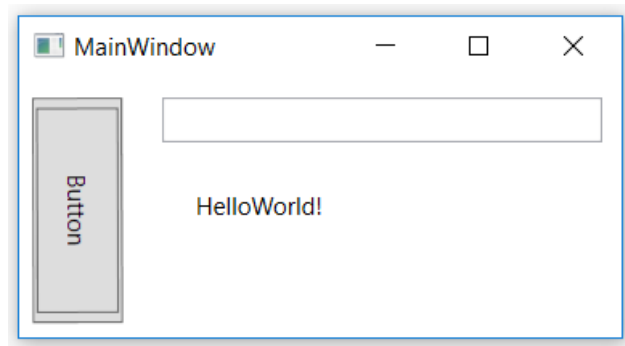
Вид в визуальном редакторе:



Пример использования:

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    label1.Content = "HelloWorld!";
}
```

Результат работы после нажатия кнопки:



Основное отличие от компонента **TextBox** в том, что **Label**, по умолчанию, нельзя использовать для ввода текста.

ListBox (текстовый список)

Свойства:

Items – коллекция строк (здесь хранится список отображаемый в компоненте)

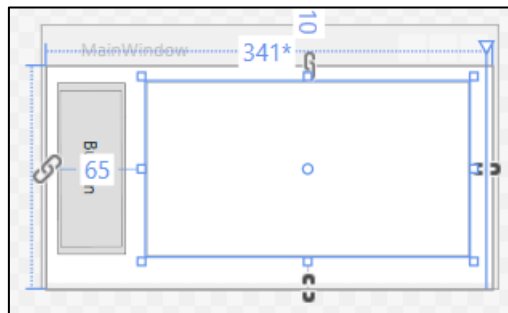
SelectedIndex – номер выбранного элемента

SelectedValue – выбранный элемент

События:

SelectionChanged – событие, генерируемое при изменении выбранного элемента

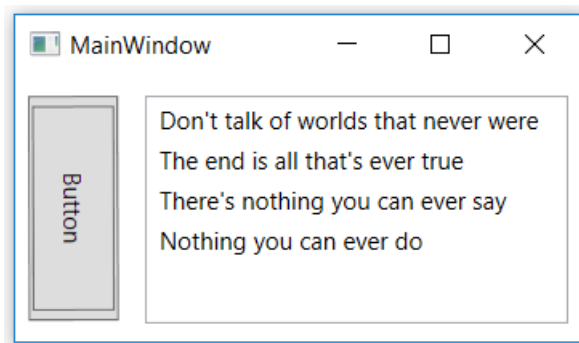
Вид в визуальном редакторе:



Пример использования:

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    lb1.Items.Add("Don't talk of worlds that never were");
    lb1.Items.Add("The end is all that's ever true");
    lb1.Items.Add("There's nothing you can ever say");
    lb1.Items.Add("Nothing you can ever do");
}
```

Результат после нажатия кнопки:



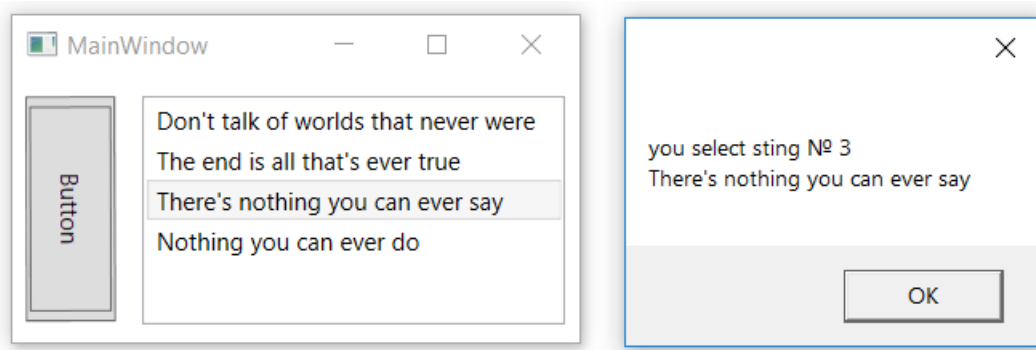
Пример реализации обработчика события SelectionChanged:

```
private void lb1_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
    //получение выбранного элемента в виде строки
    //lb1.Items - массив строк
    //lb1.SelectedIndex - номер выбранной строки
    string str = lb1.Items[lb1.SelectedIndex].ToString();
    MessageBox.Show("you select sting № " + (lb1.SelectedIndex + 1) + "\n" + str);
}
```

Альтернативный вариант:

```
private void lb1_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
    //получение выбранного элемента в виде строки
    string str = lb1.SelectedValue as string;
    MessageBox.Show("you select sting № " + (lb1.SelectedIndex + 1) + "\n" + str);
}
```


Результат работы после нажатия кнопки и выбора строки:



ComboBox (выпадающий список)

Свойства:

Items – коллекция строк (здесь хранится список отображаемый в компоненте)

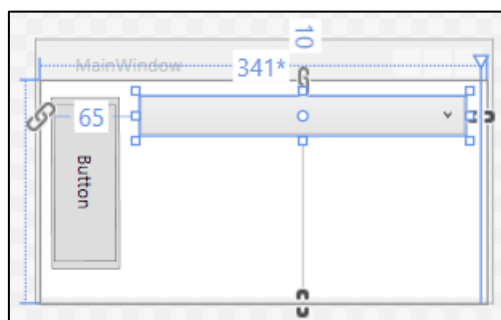
SelectedIndex – номер выбранного элемента

SelectedValue – выбранный элемент

События:

SelectionChanged – событие, генерируемое при изменении выбранного элемента

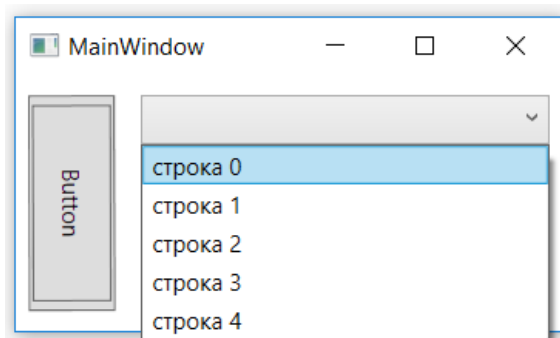
Вид в визуальном редакторе:



Пример использования:

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    for (int i = 0; i < 10; i++)
        cb1.Items.Add("строка " + i.ToString());
}
```

Результат работы после нажатия кнопки:



CheckBox (флажок)

Свойства:

Content – подпись компонента

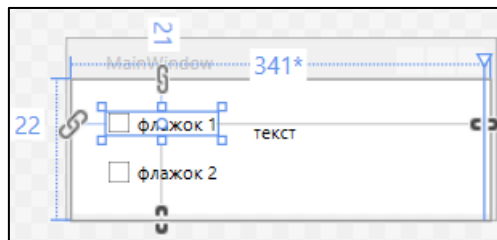
isChecked – определяет, отмечен флажок или нет

События:

Click – событие, генерируемое в момент нажатие на флажок

Checked – событие, генерируемое в момент отметки флажка

Вид в визуальном редакторе:



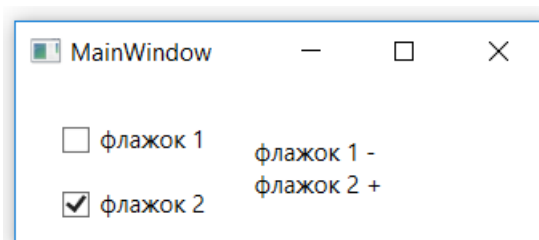
Пример использования:

```
private void CheckBox_Checked(object sender, RoutedEventArgs e)
{
    //очистка области вывода текста
    l1.Content = "";

    if (cb1.IsChecked == true)
        l1.Content += cb1.Content.ToString() + " +\n";
    else
        l1.Content += cb1.Content.ToString() + " -\n";

    if (cb2.IsChecked == true)
        l1.Content += cb2.Content.ToString() + " +\n";
    else
        l1.Content += cb2.Content.ToString() + " -\n";
}
```

Результат работы после отметки второго флажка:



Menu (меню)

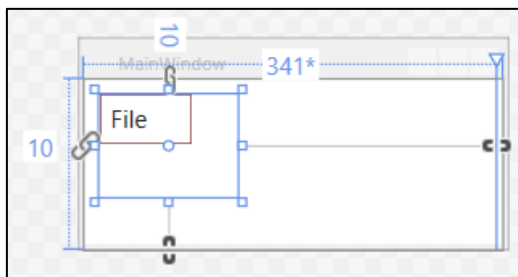
Свойства:

Header – заголовок меню

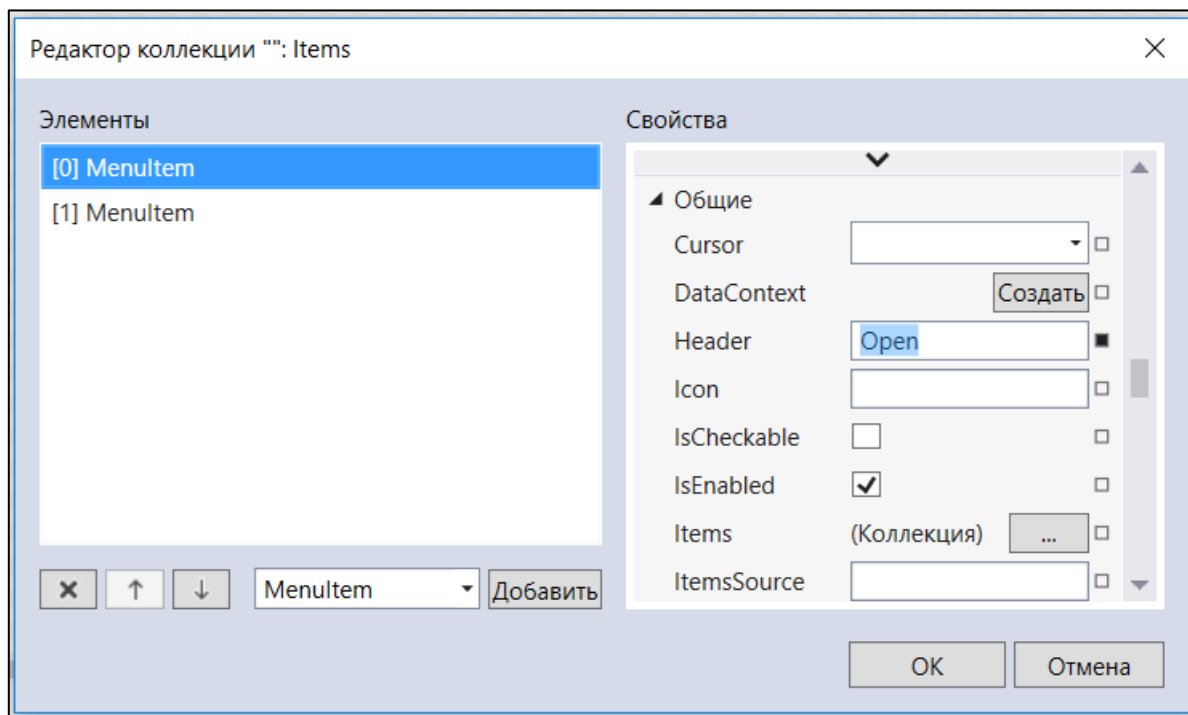
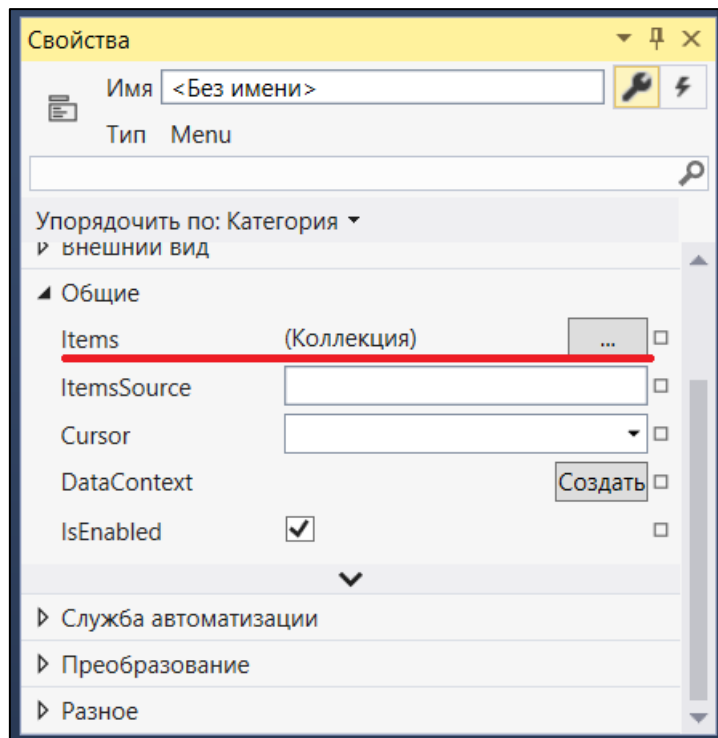
События:

Click – событие, генерируемое в момент нажатие на флажок

Вид в визуальном редакторе:



Добавить пункты меню можно через окно свойств, используя выделенную кнопку:

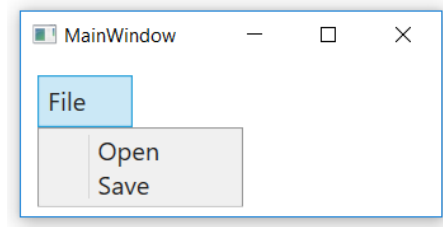


Добавить обработчик события выбора пункта меню, можно выделив соответствующий пункт меню в редакторе xaml:

```
15 <Menu HorizontalAlignment="Left" Height="67" Margin="10,10,0,0" VerticalAlignment="Top" Width="90" Background="White">
16 <MenuItem x:Name="M1" Header="File" BorderBrush="#FF8B3A3A" Height="33" Width="60" FontSize="16">
17 <MenuItem x:Name="M1open" Header="Open"/>
18 <MenuItem x:Name="M1save" Header="Save"/>
19 </MenuItem>
20 </Menu>
21
```

После чего, этот компонент будет открыт в окне **Свойства**.

Результат запуска программы:



DispatcherTimer (таймер)

Свойства:

Interval – период времени между тактами таймера в миллисекундах

Методы:

Start – запуск таймера

Stop – остановка таймера

События:

Tick – событие, генерируемое таймером каждые Interval миллисекунд

Компонент является не визуальным и создаётся в коде:

```
public partial class MainWindow : Window
{
    //создание переменной Таймер
    System.Windows.Threading.DispatcherTimer Timer;

    public MainWindow()
    {
        InitializeComponent();

        //инициализация переменной таймер
        Timer = new System.Windows.Threading.DispatcherTimer();
        //назначение обработчика события Тик
        Timer.Tick += new EventHandler(dispatcherTimer_Tick);
        //установка интервала между тиками
        //TimeSpan – переменная для хранения времени в формате часы/минуты/секунды
        Timer.Interval = new TimeSpan(0, 0, 1);
        //запуск таймера
        Timer.Start();
    }
    //обработчик события Тик
    private void dispatcherTimer_Tick(object sender, EventArgs e)
    {
        //вывод в компонент Label текущей даты и времени
        lb1.Content = DateTime.Now.Second;
    }
}
```

OpenFileDialog (диалог открытия файла)

Свойства:

DefaultExt – расширение по умолчанию

FileName – имя файла

Filter – фильтрация отображаемых файлов по расширению

Методы:

ShowDialog – показать диалог

Компонент является не визуальным и используется непосредственно в коде:

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    //создание диалога
    Microsoft.Win32.OpenFileDialog dlg = new Microsoft.Win32.OpenFileDialog();
    //настройка параметров диалога
    dlg.FileName = "Document"; // Default file name
    dlg.DefaultExt = ".txt"; // Default file extension
    dlg.Filter = "Text documents (.txt)|*.txt"; // Filter files by extension
    //вызов диалога
    dlg.ShowDialog();
    //получение выбранного имени файла
    lb1.Content = dlg.FileName;
}
```

SaveFileDialog (диалог сохранения файла)

Свойства:

DefaultExt – расширение по умолчанию

FileName – имя файла

Filter – фильтрация отображаемых файлов по расширению

Методы:

ShowDialog – показать диалог

Компонент является не визуальным и используется непосредственно в коде:

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    //создание диалога
    Microsoft.Win32.SaveFileDialog dlg = new Microsoft.Win32.SaveFileDialog();
    //настройка параметров диалога
    dlg.FileName = "Document"; // Default file name
    dlg.DefaultExt = ".txt"; // Default file extension
    dlg.Filter = "Text documents (.txt)|*.txt"; // Filter files by extension
    //вызов диалога
    dlg.ShowDialog();
    //получение выбранного имени файла
    lb1.Content = dlg.FileName;
}
```

Чтение и запись данных в текстовый файл:

Чтение данных из файла можно осуществить при помощи объекта **StreamReader**:

```
string line;

//открытие файла test.txt для чтения
System.IO.StreamReader file = new System.IO.StreamReader(@"c:\test.txt");

//построчное чтение файла
while((line = file.ReadLine()) != null)
{
    //операции над полученной строкой
}

//закрытие файла
file.Close();
```

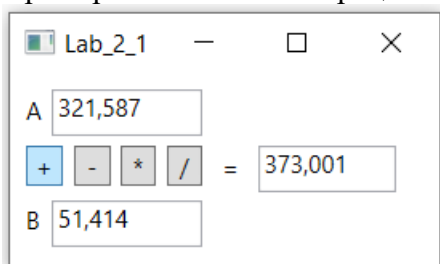
Запись данных в файл можно осуществить при помощи объекта **StreamWriter**:

```
//открытие файла test.txt для записи
using (StreamWriter outputFile = new StreamWriter(mydocpath + @"\test.txt"))
{
    //lines - массив строк
    foreach (string line in lines)
        outputFile.WriteLine(line);
}
```

Задания:

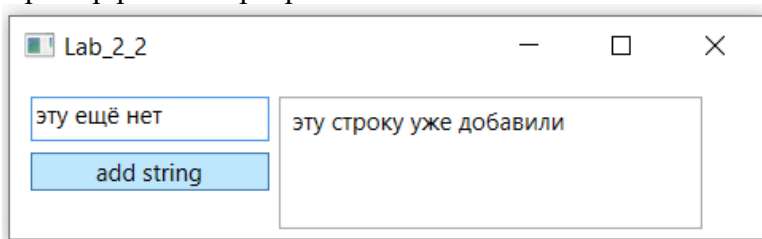
- 1) Разработайте и реализуйте приложение WPF, которое:
 - содержит 2 тестовых поля подписанных как “А” и “Б”
 - четыре кнопки подписанных как “+”, “-”, “*” и “/”
 - поле для вывода текста
 - пользователь может ввести числа в текстовые поля, нажать кнопку и получить в текстовом поле результат, соответствующей арифметической операции над введёнными числами в текстовом поле

Пример выполнения операции сложения:



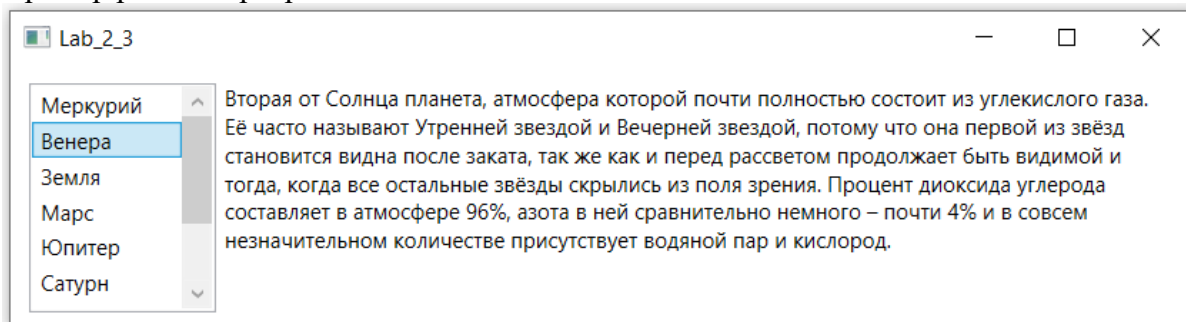
- 2) Разработайте и реализуйте приложение WPF, которое:
 - содержит текстовое поле
 - содержит текстовый список
 - содержит кнопку, подписанную как “Добавить”
 - при нажатии на кнопку, текст, записанный в текстовом поле, должен добавляться как новая строка в текстовый список, после добавления, текстовое поле должно быть очищено

Пример работы программы:



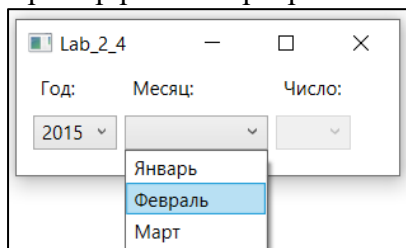
- 3) Разработайте и реализуйте приложение WPF, которое:
- содержит текстовый список, содержащий названия планет солнечной системы
 - содержит поле для вывода текста
 - при выборе названия планеты из текстового списка, выводит краткую информацию о ней в поле для вывода текста

Пример работы программы:



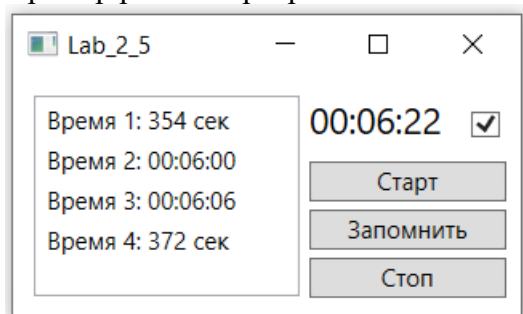
- 4) Разработайте и реализуйте приложение WPF, которое:
- содержит три выпадающих списка, с помощью которых можно выбрать год, месяц и день
 - количество дней в месяце определяется только после выбора года и месяца, до этого, выпадающий список с выбором дня должен быть не активен
 - после выбора всех трёх параметров, должно появляться сообщение с информацией о том, сколько лет, месяцев и дней прошло с выбранной даты до текущего момента

Пример работы программы:



- 5) Разработайте и реализуйте приложение WPF, которое:
- содержит поле для вывода текста, таймер, кнопки “старт”, “стоп” и “запомнить”, а также текстовый список и один флажок
 - при нажатии кнопки “старт”, в поле для вывода текста должен начинаться отсчёт секунд с момента нажатия кнопки “старт”, при нажатии кнопки “стоп”, значение счётчика секунд обнуляется, и отсчёт прекращается, при нажатии кнопки “запомнить”, текущее значение счётчика секунд добавляется в текстовый список
 - при помощи флажка, можно определить формат вывода и сохранения прошедшего времени: только секунды, либо часы + минуты + секунды

Пример работы программы:



- 6) Разработайте и реализуйте приложение WPF, которое:
- содержит меню и текстовый список
 - содержит в меню два пункта, позволяющие загрузить текст из выбранного, с помощью диалога открытия, файла текст в текстовый список, и сохранить текст из текстового списка в выбранный, при помощи диалога сохранения, файл

Список литературы:

Учебник. Создание первого приложения WPF в Visual Studio 2019:

<https://docs.microsoft.com/ru-ru/dotnet/framework/wpf/getting-started/walkthrough-my-first-wpf-desktop-application>

Более подробную информацию о компонентах можно получить в следующих источниках:

Элементы управления Windows Forms и эквивалентные элементы управления WPF
<https://msdn.microsoft.com/ru-ru/library/ms750559.aspx>