**Министерство образования и науки Российской Федерации**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ**

**УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ,**

**МЕХАНИКИ И ОПТИКИ»**

**Факультет «ИКТ»**

**Дисциплина**

«Создание программного обеспечения инфокоммуникационных систем»

**Лабораторная работа 1**

**«**Анализ предметной области. Создание проекта WPF**»**

**Выполнил:**

студент группы K3415

Дружинин К. А.

**Проверил**:

Осипов Н. А.

Санкт-Петербург

2021

**Практическое задание 1. Создание приложений WPF**

**Упражнение 1. Создание приложения Windows**

**Цель задания:**

Создать простое WPF приложение. Познакомиться с конструктором и редактором XAML. Протестировать базовые элементы пользовательского интерфейса.

**Шаги выполнения работы:**

1. Создадим новый проект WPF Application с именем «WpfHello».

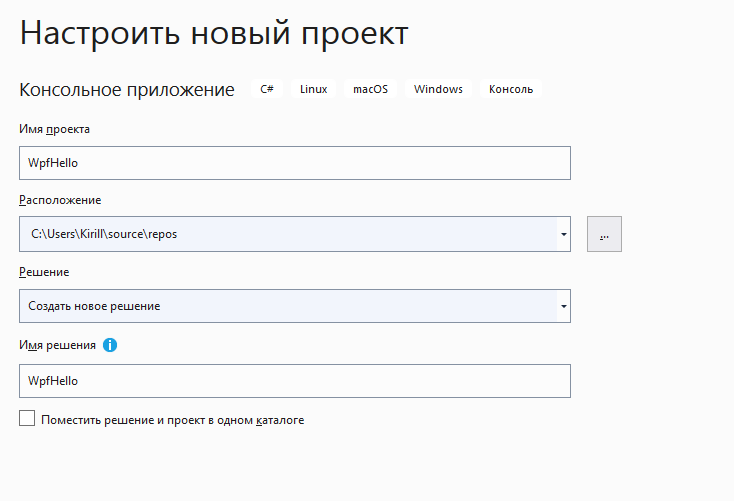


Рисунок 1 - окно создания нового приложения в Visual Studio.

1. Используя панель инструментов (Toolbox) определим на форме элементы пользовательского интерфейса **Textbox**, **Button** и **Label** в соответствии с заданием.

**Надпись Label**

Элемент управления Label (надпись) предназначен для вывода в нужном месте окна какого-либо поясняющего текста. Например, если рядом с текстовым полем (объектом типа TextBox) на форме расположить такую надпись с соответствующим текстом, то пользователю программы станет понятно, какого рода информацию надо вводить в текстовое поле.

Основным свойством элемента управления Label является свойство Text, в котором и находится поясняющая надпись. Текст, выводимый на элементе Label, может быть изменён в процессе выполнения программы. Это позволяет на объект типа Label выводить результат каких-либо вычислений, как это делается в примере в теме «Оконные приложения»:

s = a \* h / 2;

label3.Text = "s=" + s.ToString();

**Текстовое поле TextBox**

Элемент управления TextBox (текстовое поле) предназначен для ввода или вывода текстовый информации, которая записывается в свойстве Text.

Так можно использовать данный элемент для ввода данных:

double a = double.Parse(textBox1.Text);

А это пример вывода информации в текстовое поле:

int v=1;

textBox3.Text="Выбран вариант №"+v.ToString();

**Командная кнопка Button**

Элемент управления Button (командная кнопка) предназначен для выполнения какого-либо действия в программе.

Свойство Text этого элемента должно показывать назначение командной кнопки.

Если щёлкнуть мышью по этому элементу, то будет выполняться обработчик события Click, связанный с этим элементом.

Для кнопки можно задать клавиши быстрого доступа. В этом случае можно не щёлкать мышь по кнопке, а нажимать сочетание клавиш Alt+<подчёркнутый символ в надписи кнопки>. Для задания клавиши быстрого доступа необходимо в свойстве текст кнопки перед необходимой буквой поставить символ & (амперсанд).

Для того, чтобы символ & (амперсанд) воспринимался в названии просто как символ, необходимо задать свойству UseMnemonic значение false.

**Флажок CheckBox**

1. Определим свойства добавленных элементов в соответствии с заданием.



Рисунок 2 - Определенный пользовательский интерфейс в конструкторе.

1. Определим обработчики нажатия (Click) на созданные кнопки.

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

setNameButton.IsEnabled = false;

returnNameButton.IsEnabled = false;

}

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

{

try

{

System.IO.StreamWriter sw = new

System.IO.StreamWriter("username.txt");

sw.WriteLine(textBox.Text);

sw.Close();

returnNameButton.IsEnabled = true;

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

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

{

try

{

System.IO.StreamReader sr = new

System.IO.StreamReader("username.txt");

label.Content = "Приветствую Вас, уважаемый " + sr.ReadToEnd();

sr.Close();

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

private void textBox\_TextChanged(object sender, TextChangedEventArgs e)

{

setNameButton.IsEnabled = true;

}

private void button3\_Click(object sender, RoutedEventArgs e)

{

myTextBox.Text = DateTime.Now.ToString();

}

}

}

Текст 1 - определение обработчиков события Click для кнопок.

1. Проверим работу созданного приложения.

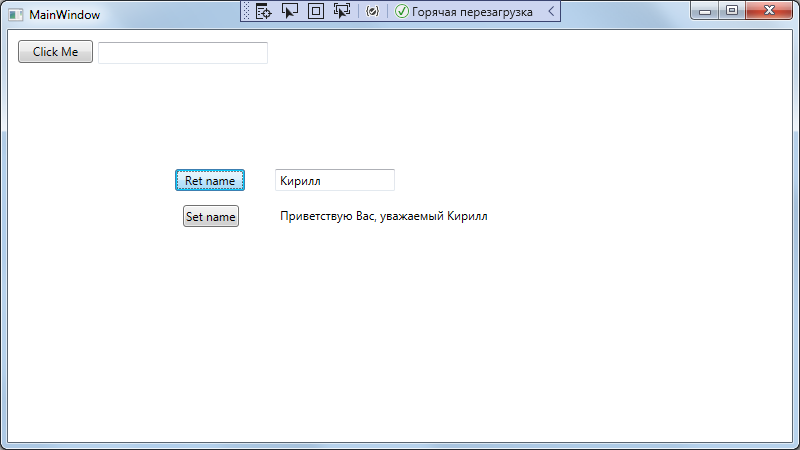


Рисунок 4 - Проверка работоспособности приложения

**Практическое задание 1. Создание приложений WPF**

**Упражнение 2. Создание навигационного приложения**

1. Создадим новый проект WPF Application с именем «WpfHelloNav».
2. В соответствии с заданием, изменим тег Window на NavigationWindow в режиме разметки XAML. Удалим тег <Grid>

<NavigationWindow x:Class="WpfHelloNav.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:WpfHelloNav"

mc:Ignorable="d"

Title="MainWindow" Height="450" Width="800" Source="/Page1.xaml">

</NavigationWindow>

Текст 2 - Редактирование формы в режиме разметки XAML.

1. Отредактируем класс формы в режиме кода, заменив базовый класс Window на NavigationWindow.

Класс NavigationWindow более или менее похож на базовый класс Window, за исключением кнопок навигации "вперед" и "назад", которые отображаются в строке сверху.

1. Запустим и протестируем приложение.

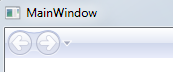


Рисунок 3 - Интерфейс приложения (NavigationWindow)

1. Добавим в приложение две страницы WPF.
2. Для главного окна установим свойство Source со значением Page1.xaml, чтобы она отображалась при запуске приложения.
3. На страницу Page1 добавим кнопку с обработчиком события Click, определенным в задании.

public partial class Page1 : Page

{

public Page1()

{

InitializeComponent();

}

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

{

Page2 page = new Page2();

this.NavigationService.Navigate(page);

}

}

}

Текст 3 – Вносимое изменение в код

1. Построим и запустим приложение, чтобы протестировать его работу.

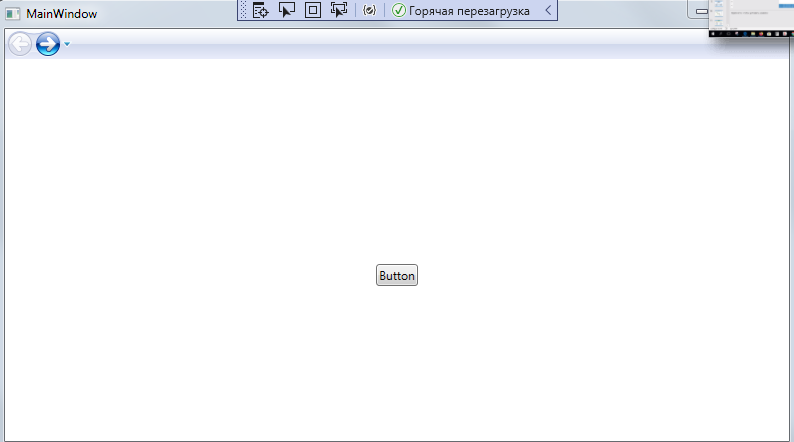


Рисунок 4 – приложение «WpfHelloNav». Страница 1

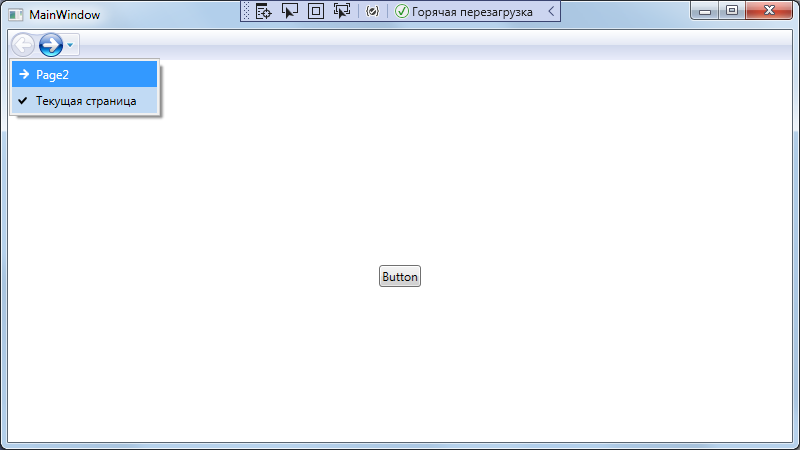


Рисунок 5 - приложение «WpfHelloNav». Страница 2

**Практическое задание 1. Создание приложений WPF**

**Упражнение 3. Реализация обработки событий**

1. Откроем проект первого упражнения.
2. Отредактируем код приложения в соответствии с заданием.
3. Добавим для текстового поля обработчик события ввода текста.
4. Добавим в обработчик события Click первой кнопки код, активирующий интерактивность второй кнопки.

namespace WpfHello

{

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

setNameButton.IsEnabled = false;

returnNameButton.IsEnabled = false;

}

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

{

try

{

System.IO.StreamWriter sw = new

System.IO.StreamWriter("username.txt");

sw.WriteLine(textBox.Text);

sw.Close();

returnNameButton.IsEnabled = true;

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

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

{

try

{

System.IO.StreamReader sr = new

System.IO.StreamReader("username.txt");

label.Content = "Приветствую Вас, уважаемый " + sr.ReadToEnd();

sr.Close();

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

private void textBox\_TextChanged(object sender, TextChangedEventArgs e)

{

setNameButton.IsEnabled = true;

}

private void button3\_Click(object sender, RoutedEventArgs e)

{

myTextBox.Text = DateTime.Now.ToString();

}

}

}

1. Текст 4 - определение обработчиков события Click для кнопок.
2. Запустим и протестируем приложение.

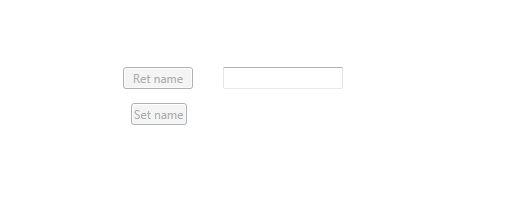


Рисунок 6 - До ввода имени кнопка задания имени неактивна.



Рисунок 7 - После начала ввода кнопка становится активна.

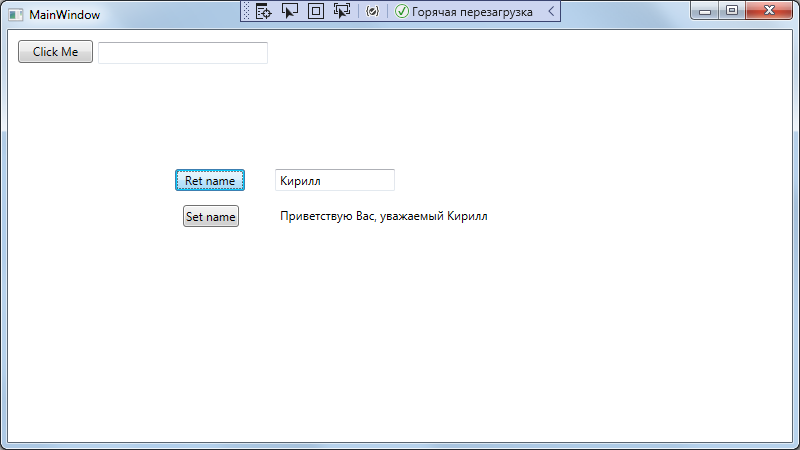


Рисунок 8 - После задания имени активируется кнопка отображения приветствия

**Практическое задание 1. Создание приложений WPF**

**Упражнение 4. Использование редактора XAML**

1. Откроем проект предыдущего упражнения.
2. Изменим корневой элемент в файле MainWindow.xaml на **Canvas**.
3. Отредактируем файл в соответствии с заданием, добавим элементы **Button** и **TextBox**.
4. Для кнопки добавим обработчик события Click и определим его в коде формы.
5. Построим и протестируем приложение. В текстовом поле отображается текущая дата и время.

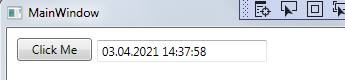


Рисунок 9 - тестирование приложения

**Практическое задание 2. Маршрутизированные события**

**Упражнение 1. Создание обработчика события**

В данном упражнении рассматривается свойство некоторых событий всплывать вверх по визуальному дереву.

Такие события будут всплывать вверх (распространяться) по визуальному дереву от исходного элемента, пока не будут обработаны или не достигнут корневого элемента. Это позволяет обработать их любому родительскому элементу, стоящему в логическом дереве выше исходного.

1. Создадим новый проект WPF Application с именем «WpfRoutedEvent\_2\_1».
2. Определим пользовательский интерфейс приложения.
3. Определим свойства добавленных элементов в соответствии с заданием.

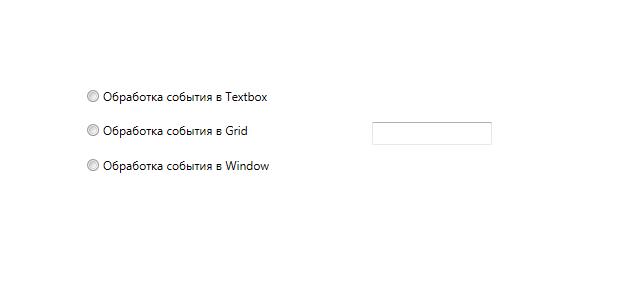


Рисунок 10 - Определенный интерфейс приложения

1. В представлении XAML зададим свойства TextChanged и TextBoxBase.TextChanged для элементов **TextBox, Grid и Window.**
2. Определим обработчики событий в классе формы.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

namespace WpfRoutedEvent\_2\_1

{

/// <summary>

/// Логика взаимодействия для MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

}

private void TextBox\_TextChanged(object sender, TextChangedEventArgs e)

{

MessageBox.Show("Event by textbox");

e.Handled = (bool)radioButtonTextbox.IsChecked;

}

private void Grid\_TextChanged(object sender, TextChangedEventArgs e)

{

MessageBox.Show("Event by Grid");

e.Handled = (bool)radioButtonGrid.IsChecked;

}

private void Window\_TextChanged(object sender, TextChangedEventArgs e)

{

MessageBox.Show("Event by Window");

e.Handled = (bool)radioButtonWindow.IsChecked;

}

}

}

Текст 5 - Обработчики.

1. Построим и протестируем приложение.

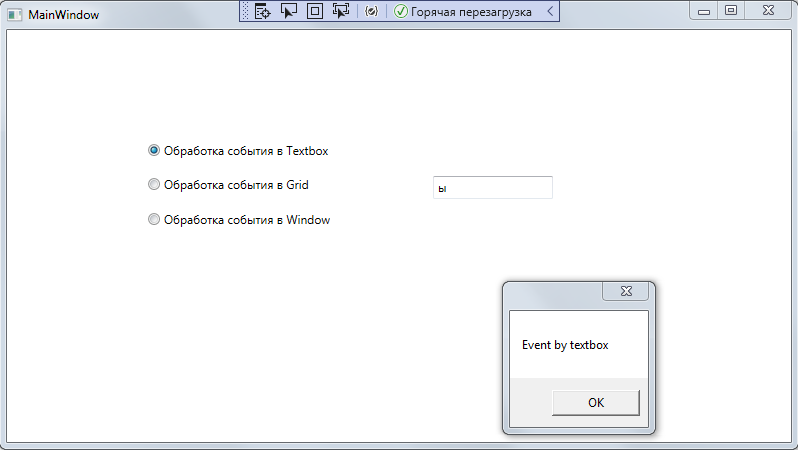


Рисунок 11 - Обработка события обработчиком элемента TextBox

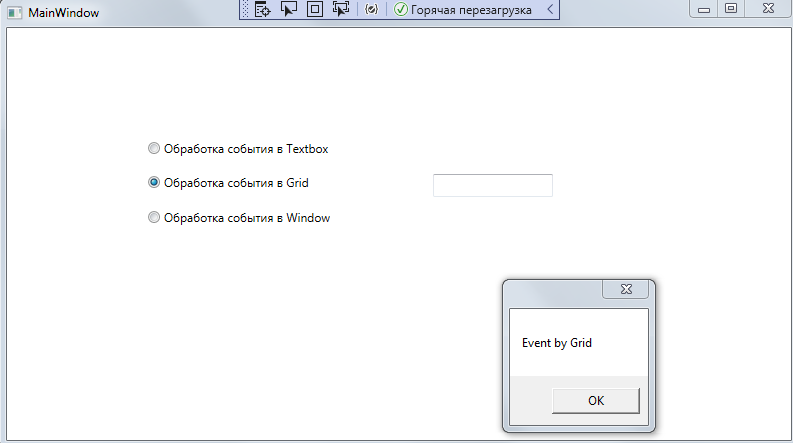


Рисунок 12 - Обработка события обработчиком элемента Grid

**Практическое задание 2. Маршрутизированные события**

**Упражнение 2. Использование маршрутизированных событий**

1. Создадим новый проект WPF Application с именем «WpfRoutedEvent\_2\_2».
2. Определим пользовательский интерфейс приложения.



Рисунок 13 - пользовательский интерфейс приложения.

1. Определим свойства добавленных элементов в соответствии с заданием.
2. Добавим обработчик события клика по кнопке на элемент StackPanel, содержащий добавленные кнопки.

namespace WpfRoutedEvent\_2\_2

{

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

}

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

{

FrameworkElement feSource = e.Source as FrameworkElement;

double a = Double.Parse(textBox.Text);

switch (feSource.Name)

{

case "buttonAdd":

a += a;

break;

case "buttonMultiplex":

a \*= a;

break;

}

e.Handled = true;

textBox.Text = String.Format("{0:#.##}", a);

}

}

}

Текст 6 - Обработчик события клика по кнопке на элемент StackPanel.

1. Построим и протестируем приложение.

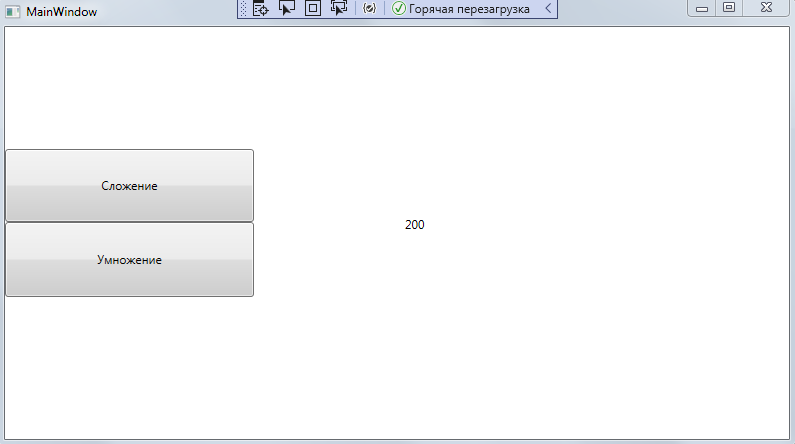


Рисунок 14 - Состояние приложения до нажатия кнопок.

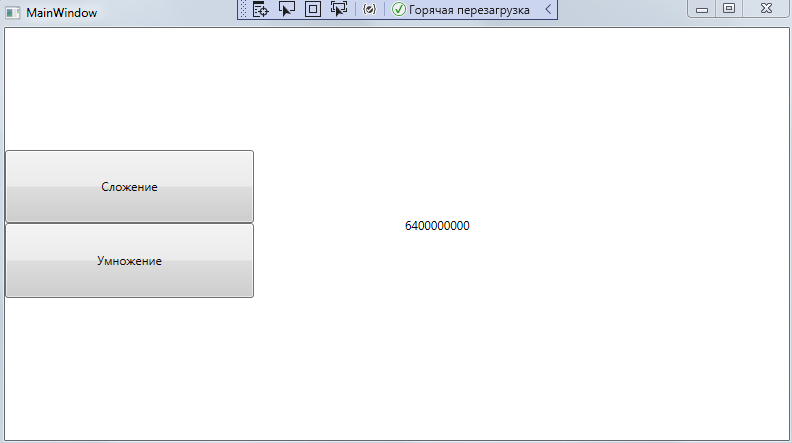


Рисунок 15 - Состояние приложения после умножения.

**Практическое задание 3. Работа с командами**

**Упражнение 1. Применение встроенных команд**

1. Создадим новый проект WPF Application с именем «NCommand».
2. Определим пользовательский интерфейс приложения.

<Window x:Class="NCommand.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:NCommand"

mc:Ignorable="d"

Title="MainWindow" Height="450" Width="800">

<Grid>

<Menu RenderTransformOrigin="0.5,0.5" VerticalAlignment="Top">

<MenuItem Header="Правка">

<MenuItem Header="Копировать" CommandTarget="{Binding ElementName=MainWindow}"

Command="Copy"/>

<MenuItem Header="Вырезать" CommandTarget="{Binding ElementName=MainWindow}"

Command="Cut"/>

<MenuItem Header="Вставить" CommandTarget="{Binding ElementName=MainWindow}"

Command="Paste"/>

</MenuItem>

</Menu>

</Grid>

</Window>

Текст 7 - пользовательский интерфейс приложения.

1. Определим свойства добавленных элементов в соответствии с заданием.
2. Определим пункты меню и зададим для них соответствующие команды.

namespace NCommand

{

class CustomCommands

{

public static RoutedUICommand Launch { get; }

}

Текст 8 – Вносимое изменение.

1. Добавим элементы пользовательского интерфейса TextBox и RichTextBox.

<TextBox x:Name="textBox" HorizontalAlignment="Left" Margin="10,46,0,0" Text="TextBox" TextWrapping="Wrap" VerticalAlignment="Top" Width="120"/>

<RichTextBox x:Name="richTextBox" Margin="10,79,10,10">

<FlowDocument>

<Paragraph>

<Run Text="RichTextBox"/>

</Paragraph>

</FlowDocument>

</RichTextBox>

<CheckBox x:Name="checkBox" Content="Enable Launch Command" HorizontalAlignment="Left" Margin="152,46,0,0" VerticalAlignment="Top"/>

<Button Command="local:CustomCommands.Launch" Content="Launch" HorizontalAlignment="Left" Margin="683,40,0,0" VerticalAlignment="Top" Width="93"/>

Текст 9 – Вносимое изменение.

**Практическое задание 3. Работа с командами**

**Упражнение 2. Создание и использование настраиваемой команды**

1. Откроем проект предыдущего упражнения.
2. Добавим в проект новый класс – CustomCommands.
3. static CustomCommands()

Текст 10 – Вносимое изменение.

1. Отредактируем ранее созданный класс CustomCommands в соответствии с заданием.

static CustomCommands()

{

InputGestureCollection myInputGestures = new InputGestureCollection();

myInputGestures.Add(new KeyGesture(Key.L, ModifierKeys.Control));

Launch = new RoutedUICommand("Запуск", "Launch", typeof(CustomCommands), myInputGestures);

}

}

Текст 11 – Вносимое изменение.

1. Отредактируем элемент меню, добавив новый элемент MenuItem с заголовком «Launch».
2. <MenuItem Header="Menu Items">
3. <MenuItem Header="Launch" Command="local:CustomCommands.Launch"/>
4. </MenuItem>

Текст 12 – Вносимое изменение.

1. Отредактируем пользовательский интерфейс, добавив элементы Button и CheckBox со свойствами, определенными в условии лабораторной работы.

<Window x:Class="NCommand.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:NCommand"

mc:Ignorable="d"

Title="MainWindow" Height="450" Width="800">

<Grid>

<Menu RenderTransformOrigin="0.5,0.5" VerticalAlignment="Top">

<MenuItem Header="Правка">

<MenuItem Header="Копировать" CommandTarget="{Binding ElementName=MainWindow}"

Command="Copy"/>

<MenuItem Header="Вырезать" CommandTarget="{Binding ElementName=MainWindow}"

Command="Cut"/>

<MenuItem Header="Вставить" CommandTarget="{Binding ElementName=MainWindow}"

Command="Paste"/>

</MenuItem>

<MenuItem Header="Menu Items">

<MenuItem Header="Launch" Command="local:CustomCommands.Launch"/>

</MenuItem>

</Menu>

<TextBox x:Name="textBox" HorizontalAlignment="Left" Margin="10,46,0,0" Text="TextBox" TextWrapping="Wrap" VerticalAlignment="Top" Width="120"/>

<RichTextBox x:Name="richTextBox" Margin="10,79,10,10">

<FlowDocument>

<Paragraph>

<Run Text="RichTextBox"/>

</Paragraph>

</FlowDocument>

</RichTextBox>

<CheckBox x:Name="checkBox" Content="Enable Launch Command" HorizontalAlignment="Left" Margin="152,46,0,0" VerticalAlignment="Top"/>

<Button Command="local:CustomCommands.Launch" Content="Launch" HorizontalAlignment="Left" Margin="683,40,0,0" VerticalAlignment="Top" Width="93"/>

</Grid>

</Window>

Текст 13 – Вносимое изменение.

1. Отредактируем класс формы, добавив привязку CommandBinding для команды Launch, обработчики событий Launch\_Handler и LaunchEnabled\_Handler.

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

CommandBinding abinding = new CommandBinding();

abinding.Command = CustomCommands.Launch;

abinding.Executed += new ExecutedRoutedEventHandler(Launch\_Handler);

abinding.CanExecute += new CanExecuteRoutedEventHandler(LaunchEnabled\_Handler);

this.CommandBindings.Add(abinding);

}

private void Launch\_Handler(object sender, ExecutedRoutedEventArgs e)

{

richTextBox.AppendText(textBox.Text);

using (System.IO.StreamWriter writer = new System.IO.StreamWriter("log.txt", true))

{

writer.WriteLine("Внесено {0}: {1} ", textBox.Text,

DateTime.Now.ToShortDateString() + ", время: " +

DateTime.Now.ToLongTimeString());

writer.Flush();

}

}

private void LaunchEnabled\_Handler(object sender, CanExecuteRoutedEventArgs e)

{

e.CanExecute = (bool)checkBox.IsChecked;

}

}

}

Текст 14 – Вносимое изменение.

1. Соберем и протестируем приложение.

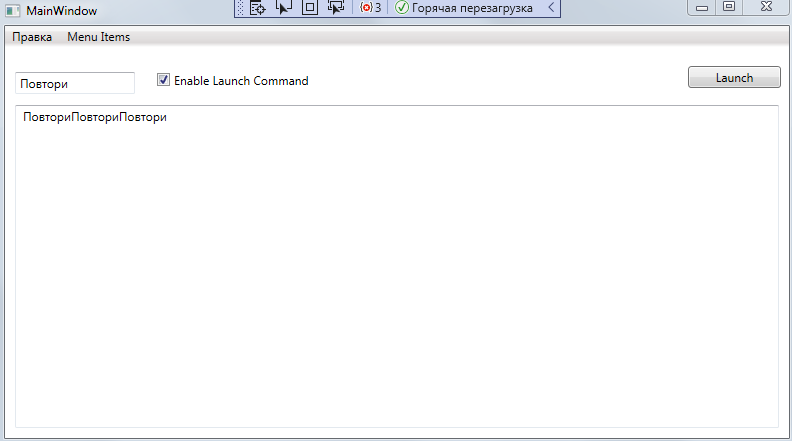


Рисунок 16 - Тестирование приложения

**Вывод**

В данной лабораторной работе были рассмотрены:

* Процесс создания WPF приложения в среде разработки Visual Studio
* Работа с элементами пользовательского интерфейса, их создание и настройка посредством возможностей Toolbox, а также непосредственно через редактирование XAML файлов проекта.

Изучение Label TextBox и Button Checkbox

* Работа с событиями в WPF приложении.
* Работа с командами, использование встроенных команд и создание новых.
* Написание простых функций обработки

## Изученые элементы:

## Ключевым элементом в системе графического интерфейса в WPF является окно, которое содержит все необходимые элементы управления. Окно в WPF представлено классом Window, который является производным от класса ContentControl. Поэтому окно является элементом управления содержимым, и как, к примеру, кнопка, может содержать в себе один дочерний элемент.

## *Надпись Label*

Элемент управления **Label** (надпись) предназначен для вывода в нужном месте окна какого-либо поясняющего текста. Например, если рядом с текстовым полем (объектом типа **TextBox**) на форме расположить такую надпись с соответствующим текстом, то пользователю программы станет понятно, какого рода информацию надо вводить в текстовое поле.

Основным свойством элемента управления **Label** является свойство **Text**, в котором и находится поясняющая надпись. Текст, выводимый на элементе **Label**, может быть изменён в процессе выполнения программы. Это позволяет на объект типа **Label** выводить результат каких-либо вычислений, как это делается в примере в теме «Оконные приложения»:

***Текстовое поле TextBox***

Элемент управления **TextBox** (текстовое поле) предназначен для ввода или вывода текстовый информации, которая записывается в свойстве **Text**.

***Командная кнопка Button***

Элемент управления **Button** (командная кнопка) предназначен для выполнения какого-либо действия в программе.

Свойство **Text** этого элемента должно показывать назначение командной кнопки.

Если щёлкнуть мышью по этому элементу, то будет выполняться обработчик события **Click**, связанный с этим элементом.

***Флажок CheckBox***

Элемент управления флажок применяется для включения или выключения какого-либо режима работы программы. Для проверки состояния флажка используется его свойство **Checked** (**true** — включен, **false** — выключен).

В свойстве **Text** даётся пояснение по работе флажка.

Чаще всего флажки используют в окнах диалога. Флажки можно применять подиночке или группами, причём состояние флажков в группах устанавливается независимо друг от друга.

Знакомство с языком разметки  XAML, чтобы предоставить декларативную модель для программирования приложений.