

# Упражнение №3 по ПС - Част 1

## WPF

*Visual Studio и GUI приложения*  
*Основи на XAML*  
*System.Windows*  
*Window*  
*Контроли*  
*Събития*

### Целта на това упражнение:

Запознаване с основните принципи и класове, които осигуряват функционалността на GUI рамката WPF.

### Необходими познания:

От предметите дотук и предното упражнение трябва да можете:

- Да имате основни умения с езика C#
- Да може да ползвате .Net класове.
- Понятие за Markup езици.
- Да можете да разписвате класове, наследяващи други класове.

### Легенда:

Най-важното

- Схематично и накратко

Обяснение. Уточнение.

**Пример:** или **Задача:** оградено трябва да изпълните за да е синхронизиран проекта ви

- Подходящо място да стартирате изпълнение за да проверите дали работи вярно.

**Задачите в упражнението изграждат:**

Университетска информационна система.

В това упражнение: Създаваме ново приложение с графичен интерфейс

- създаваме интерфейса на прозореца, който служи за въвеждане/извеждане на данните на студент
- създаваме функционалност за блокиране и активиране, попълване и изтриване на контролите
- използваме функционалността от предните два проекта в нови графичен интерфейс (използваме класа Student, за да попълваме данни от него)

**В края на упражнението:**

Потребителят вижда интерфейс за въвеждане и извеждане на данни за студент.

Интерфейсът има готова функционалност за показване и скриване на данните за студент.

Проектът е настроен да използва функционалност от другите два проекта.

## 1. Още малко C#

### 1.1. Partial Class

В C# **partial** позволява тялото на един клас да бъде разделено в няколко файла.

```
partial class Employee
{
    public void SomeFunc()
    {
        ...
    }
    ...
}
```

- Модификаторът **partial** е приложим също и за структури, интерфейси, методи.

Нас в момента ни вълнуват основно приложението за класове.

**Пример:** Следните два класа са еквивалентни:

```
public class MyClass
{
    public MyPartialClass()
    {
    }

    public void Method1(int val)
    {
        Console.WriteLine(val);
    }



    public void Method2(int val)
    {
        Console.WriteLine(val);
    }
}
```

```
public partial class MyClass
{
    public MyPartialClass()
    {
    }

    public void Method1(int val)
    {
        Console.WriteLine(val);
    }
}

public partial class MyClass
{
    public void Method2(int val)
    {
        Console.WriteLine(val);
    }
}
```

**Задача:** Добавяме нов проект към решението (solution-a):

- ("File" → "Add" → "New Project ...")  
(Или от "Solution Explorer" → "Solution" →  → "Add" → "New Project ...")
- В диалоговия прозорец изберете "Visual C#" → "Windows" → "WPF app"  
(За Visual Studio 2017: "Visual C#" → "Windows Classic Desktop" → "WPF app")
- Кръстете новия проект **StudentInfoSystem**.
- В Solution Explorer с десен бутон на новия проект изберете "Set as StartUp Project"  
("Solution Explorer" → "StudentInfoSystem" →  → "Set as StartUp Project")

- Името на проекта ще стане удебелено.

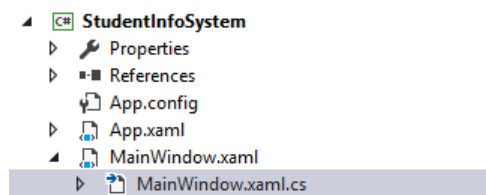
При наличие на повече от един проект Visual Studio трябва да знае кой да стартира при натискане на бутона Start (F5). Ще стартира StartUp проекта.

Добра идея е да затворите всички файлове отворени за редакция от другия проект, за да не става объркване.

Ако ви липсват тези опции в диалоговият прозорец за нов проект:

<https://stackoverflow.com/questions/41189398/no-templates-in-visual-studio-2017>

**Задача:** Разгледайте файла MainWindow.xaml.cs.



- В него Visual Studio е създадо `partial class MainWindow`
- Това означава, че в този файл е само част от класа.

Къде е другата част? Ще разберем малко по-надолу.

## 1.2. Public Class

Публичните класове са видими от други асемблита (проекти).

- Всички деклариран типове от различни имена пространства (namespace) са видими в рамките на същия проект.
- Извън проекта се виждат само онези типове, които са деклариран като публични.

```
static public class Class
{
}

```

В проектите **UserLogin** и **StudentRepository**:

**Задача:** Обявете за публични всички класове, които бихме използвали от визуалното приложение: `Logger`, `LoginValidation`, `User`, `UserData`... и други.  
(Класовете Program от двете конзолни приложения няма нужда да са публични)

## 2. Още малко Visual Studio

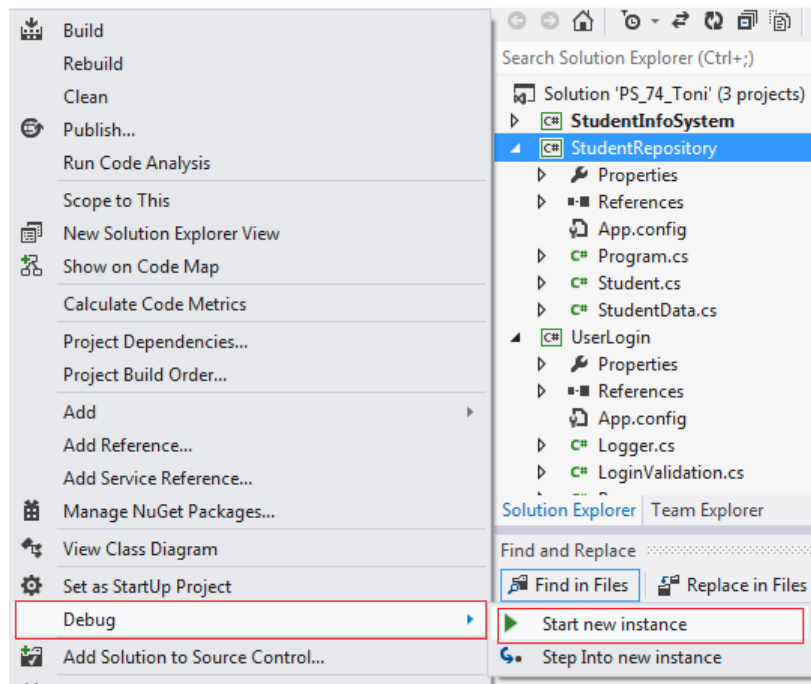
### 2.1. Работа с повече от един проекти

Едно решение (solution) може да се състои от множество проекти.

### 2.1.1. Стартиране на проект

При стартиране на дебъг или на самото приложение, Visual Studio стартира само един от проектите - маркирания като StartUp проект.

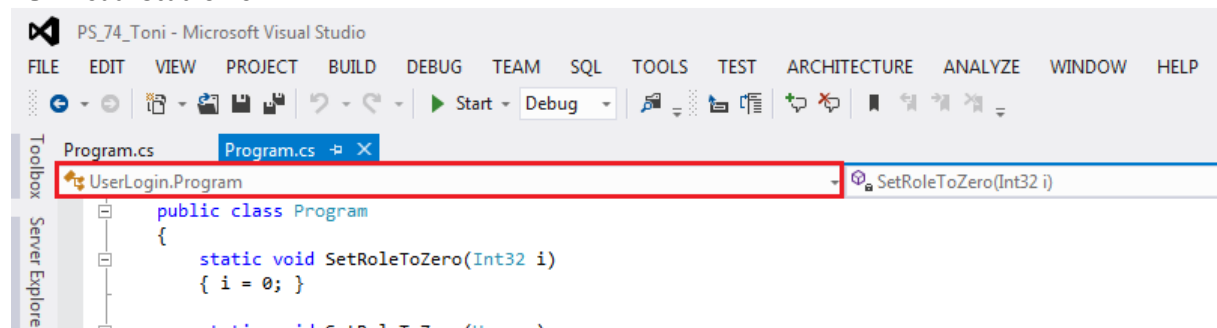
Може да стартирате и конкретен проект, без да променяте кой е StartUp проекта, с:  
Десен бутон на желанния проект и после „Debug” → “Start new instance”



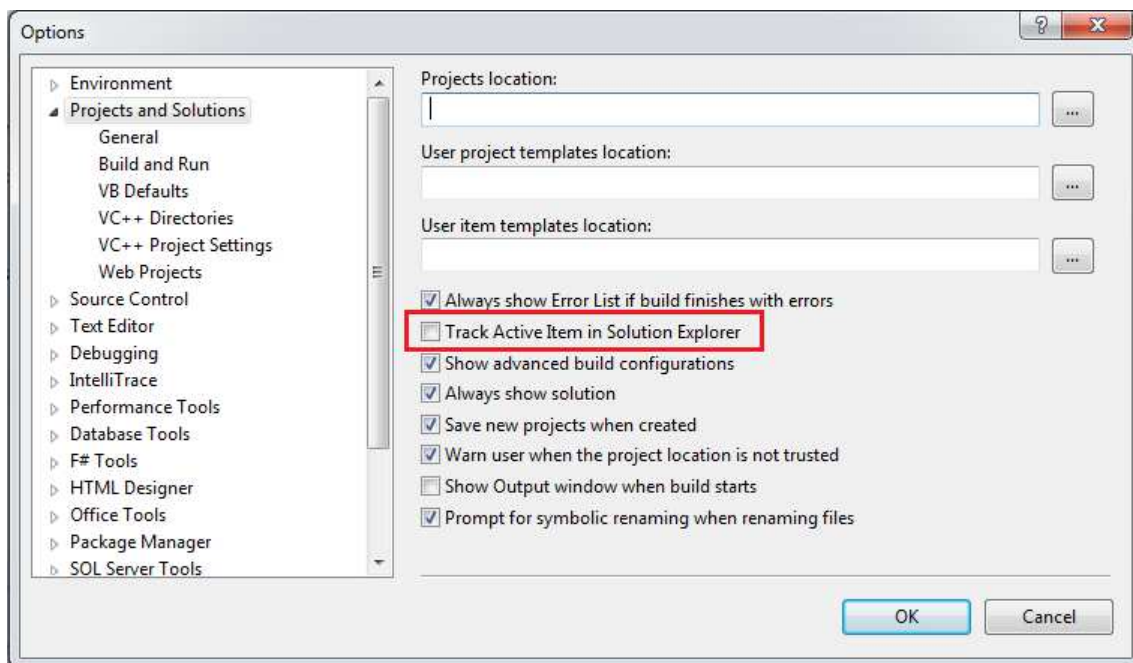
### 2.1.2. Ориентация в кода

Може да се ориентирате текущият отворен файл за редакция на кой проект принадлежи по имменното пространство в него. Това можете да видите бързо най-отгоре на екрана за редакция.

Във Visual Studio 2012:

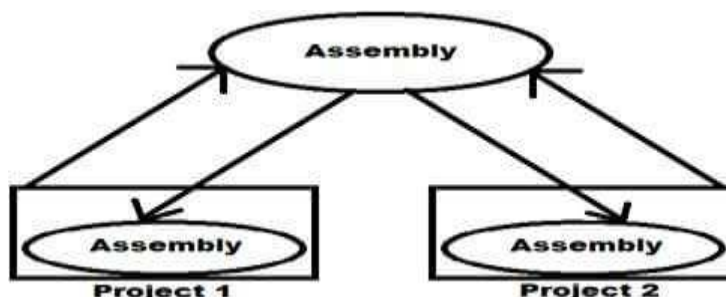


Друга настройка, която може да помогне, позволява при преминаване на нов файл в раздела за редакция да се маркира в дървото на Solution Explorer.  
("Tools" → "Options" → "Projects and Solutions")




### 2.1.3. Преизползване на функционалност

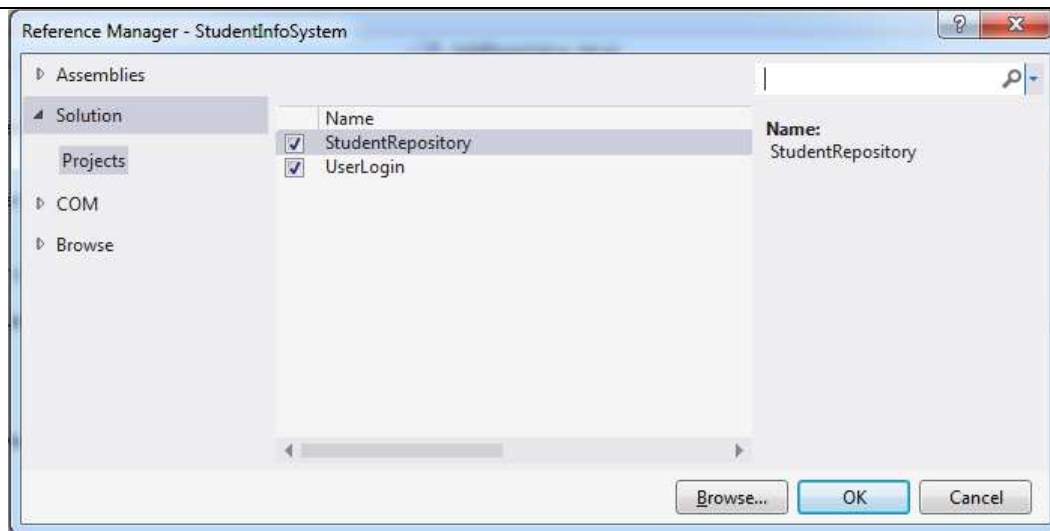
Всеки проект се компилира до отделно (в асембли), т.е. може да се използва функционалността му в друг проект.



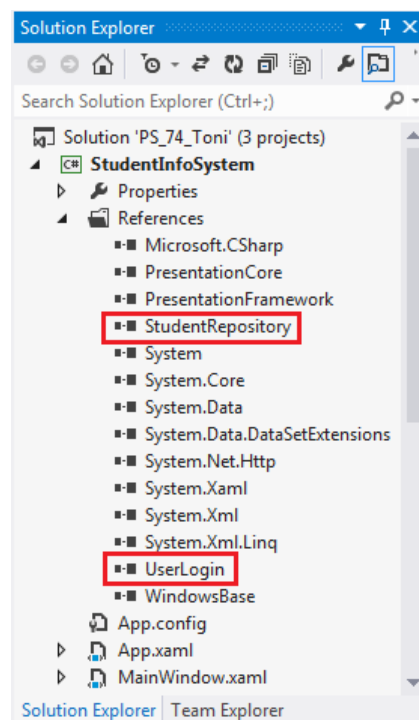
В проекта **StudentInfoSystem**:

**Пример:** Да добавим към новия проект функционалността от проектите UserLogin и StudentRepository, за да може да използваме декларираните в тях типове:

- “Solution Explorer” → “StudentInfoSystem” →  → “Add Reference...”
- В отвореният се диалогов прозорец: “Solution” → “Projects” и избирате двата проекта да бъдат вкл/чени като библиотеки към проекта StudentInfoSystem.



- Добавените проекти ще се появят в папката References към вече участващите библиотеки, необходими за реализацията на WPF приложение:



- Сега вече може да използваме всички публични типове от UserLogin и StudentRepository във файловете от StudentInfoSystem с пълно име или като добавим необходимите using клаузи:

```
using UserLogin;
using StudentRepository;
```

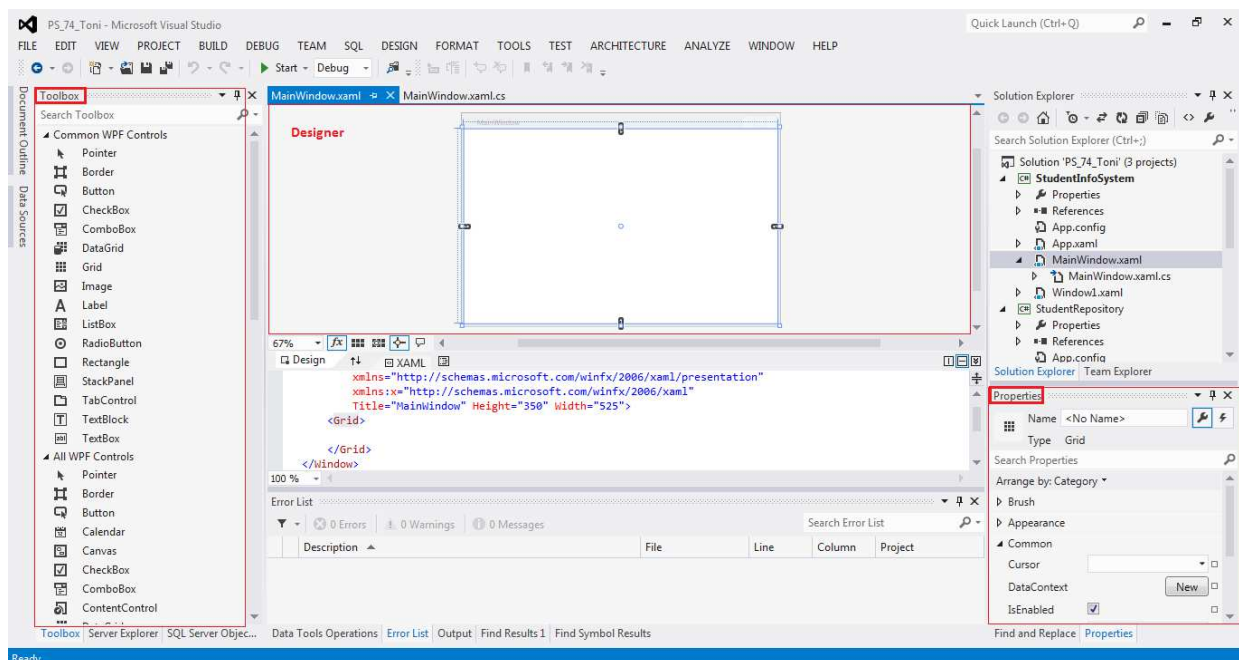
В проекта **StudentRepository**:

**Задача:** Да използваме класа User от UserLogin в проекта StudentRepository:

- Добавете UserLogin като reference към StudentRepository
- Създайте нов клас StudentValidation към проекта StudentRepository
- В класа StudentValidation функция GetStudentDataByUser, която:
  - Да приема обект от тип User
  - Да връща обект от тип Student
  - Ако в подадения обект от тип User не е посочен факултетен номер или не е открит студент по попълнения факултетен номер, да уведомява по някакъв начин извикващият функцията.

## 2.2. Работа с визуални приложения

След като е отворен един WPF проект вашето Visual Studio трябва да има приблизително този вид:



Новите прозорци (Designer, ToolBox и Properties) са свързани с разработката с графичен интерфейс:

- Designer – предлага предварителен преглед на интерфейса без да с стартира приложението, а също и позволява промяна на всеки един от елементите, които ще се визуализират.  
Дизайнерът се отваря на мястото, където се отваря редакторът за код при файловете с код.
- ToolBox – списък с всички готови елементи, които могат да се поставят в прозорци и ще се визуализират.
- Properties – при селекция на поставен елемент за визуализиране, изписва настройките на селектирания елемент.

## 3. Въведение в WPF



WPF = Windows Presentation Foundation

WPF е система за разработката на клиентски Windows приложения, позволяваща удивително потребителско изживяване. С WPF могат да се създават широк набор от самостоятелни или базирани на брауъра приложения.

Ядрото на WPF е независимо от резолюцията и векторно базирано, така че да се възползва от модерния графичен хардуер. WPF дава достъп до обширен набор от функции за разработка на приложения, включващи Extensible Application Markup Language (XAML), контроли (controls), връзка с данните (data binding), 2D и 3D графика, анимация, стилове, шаблони, документи, медия, текст и текстоформление. WPF е включен в .NET Framework и е възможно в него да се вграждат и други елементи от библиотеките с класове на .NET.

Програмирането с WPF включва разглежданите в предното упражнение дейности като: инстанциране на класове, настройка на свойства, извикване на методи и прихващане на събития, използвайки програмен език C# или Visual Basic.

### 3.1. Markup и Code-Behind

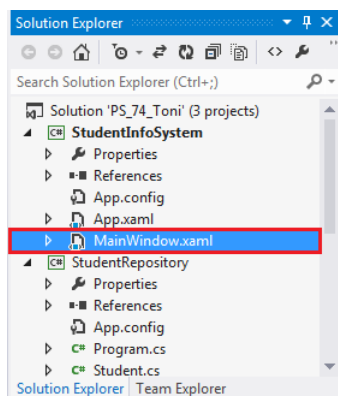
Markup = описанието на интерфейса посредством XAML.


Code-Behind = свързания код файл към интерфейса, който реализира логика (в нашия случай на C#)

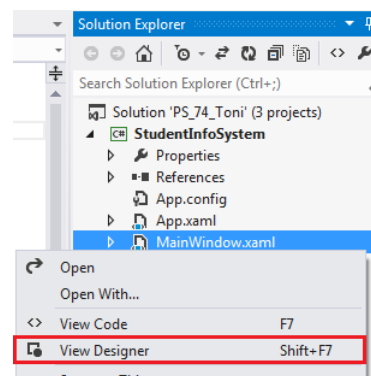
#### 3.1.1. Markup


Markup се съхранява във файл с разширение .xaml  
Отваря се по един от следните 3 начина:

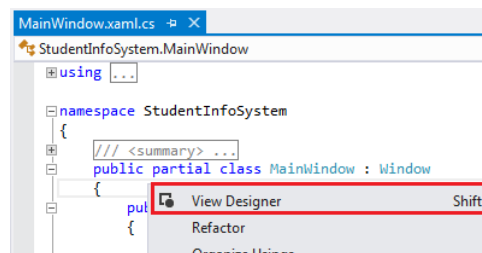
Двоен клик на .xaml файл:



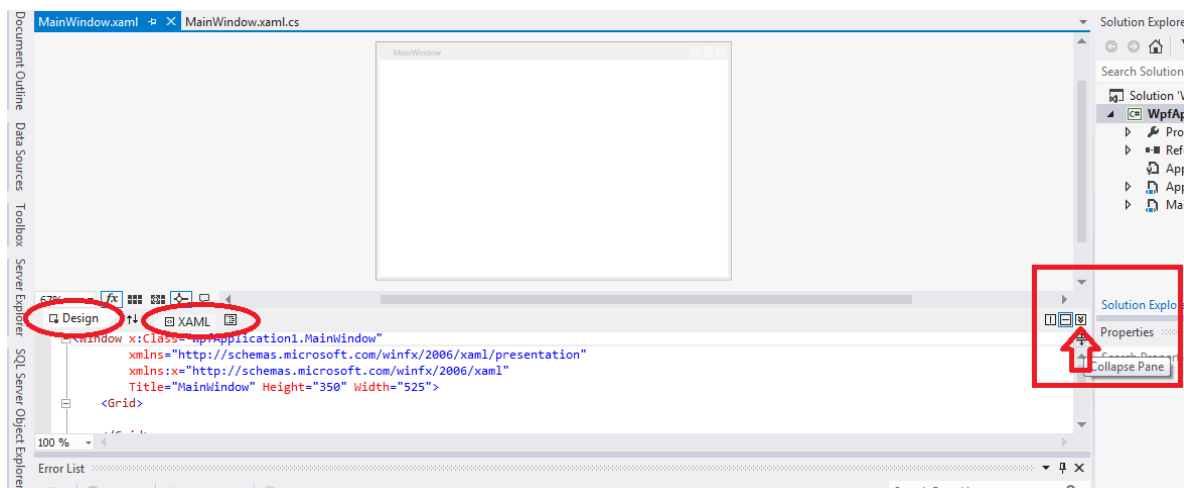
.xaml файл →  → "View Designer"



Code-Behind →  → "View Designer"



По подразбиране Visual Studio извежда екран разделен между дизайнера и XAML кода, когато файлът описва прозорец (форма).



Редакция в единия раздел автоматично се отразява и в другия.

В **долния десън ъгъл** може да определяте подредбата на разделите.

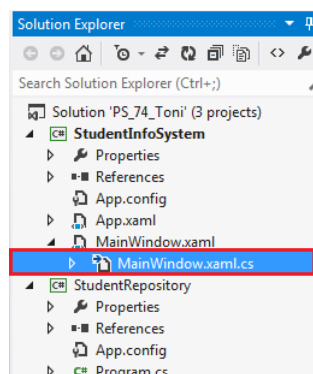
Ако решите да скриете някои от тях, ще се визуализира само единия и ще трябва да избирате кой раздел от бутоните в **долния ляв ъгъл**.

### 3.1.2. Code-Behind

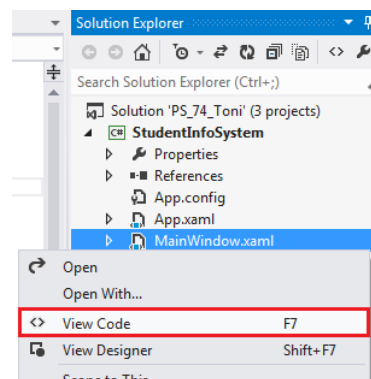
Code-Behind се съхранява във файл с разширение .cs

Отваря се по един от следните 3 начина:

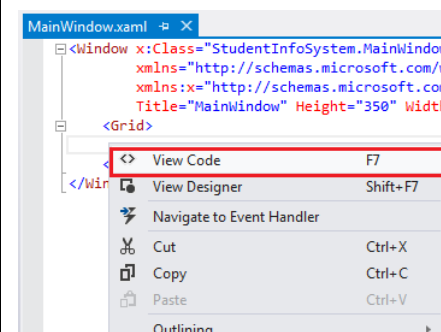
Разпъване на .xaml файла,  
и двоен клик на .xaml.cs:



.xaml файл → → "View Code"



Markup → → "View Code"



Markup и Code-Behind заедно описват един интерфейс (обикновено прозорец (форма)).

## 3.2. Езикът XAML

XAML = eXtensible Application Markup Language

- Език за описване на графични интерфейс.
- Генерира се атоматично от дизайнера на Visual Studio.

- Може и да се редактира директно от програмиста или дизайнера.

XAML е маркъп (тагов) език, подобно на HTML, XML, ...

Вашият току що създаден проект се състои от няколко файла, един от тях е: *MainWindow.xaml*. Това е началния прозорец на приложението, този, който се показва при стартиране му, освен ако изрично не зададете нещо друго.

XAML кодът, който ще намерите в него, трябва да изглежда така:



```
<Window x:Class="StudentInfoSystem.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="350" Width="525">
    <Grid>

    </Grid>
</Window>
```

Това е основния XAML, който Visual Studio създава за вашия прозорец.

XAML е език за програмиране.

Създадените обекти в него, макар и с тагове, са валидни обекти на .Net класове.

### 3.2.1. Маркър на един празен XAML файл

В MainWindow.xaml се намират два елемента (обекта) – Window и Grid описани с два тага, всеки с отварящ и затварящ таг:

```
<Window></Window>
<Grid></Grid>
```

Елементът Grid е вложен в Window, т.е. тагът Grid се намира между отварящият и затварящ таг на Window, т.е. Grid е член на Window:

```
<Window>
    <Grid></Grid>
</Window>
```

Елементът (обектът) Window има зададени стойности на множество свойства:

```
x:Class
xmlns
xmlns:x
Title
Height
Width
```

Свойствата се настройват в отварящият таг:

```
<Window x:Class= ...
        xmlns=...
        xmlns:x=...
```

```
Title=...
Height=...
Width=...>

</Window>
```

Свойството `x:Class` ще разгледаме в следващата точка.

```
x:Class="StudentInfoSystem.MainWindow"
```

Свойствата `xmlns` и `xmlns:x` указват версията на шаблона, по който да се интерпретира текущия файл. (Като цяло нямаме работа с тях).

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

Свойствата `Title`, `Height` и `Width` задават размера и заглавието на прозорецът, който ще се визуализира в резултат от това описание.

```
Title="MainWindow" Height="350" Width="525"
```

Елементът (обектът) `Grid` представлява таблица. Не са му зададени никакви свойства или настройки, следователно представлява таблица само с един ред и една колона, обхващаща целият прозорец.

```
<Grid>
</Grid>
```

Цялото видимо съдържание на прозорецът трябва да се намира (всички тагове трябва да са вложени в) `Grid`-а.

```
<Grid>
  <!-- Съдържание тук -->
</Grid>
```

*В момента няма нищо вътре в таблицата `Grid`, т.е. прозорецът се визуализира празен.*

### 3.3. Класът `Window`

- `Window` е клас в `.Net`, с готова функционалност, който реализира изрисването и поведението на прозорец на екрана.
- В нашите проекти създаваме класове, наследяващи класа `Window` напр. `MainWindow`
- В нашия наследяващ клас (`MainWindow`) добавяме полета и функционалност, в зависимост от това какво трябва да съдържа (да се показва в) прозореца.

Наследяващият клас (`MainWindow`) е дефиниран в два файла и на два езика:

1. В Markup файла (`MainWindow.xaml`) на XAML
2. В Code-Behind файла (`MainWindow.xaml.cs`) на C#

**Пример:** Двете части на класа `MainWindow`:

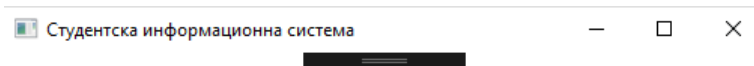
<pre>&lt;Window x:Class="StudentInfoSystem.MainWindow"         xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"         xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"         Title="MainWindow" Height="350" Width="525"&gt;     &lt;Grid&gt;      &lt;/Grid&gt; &lt;/Window&gt;</pre>	<pre>public partial class MainWindow : Window {     public MainWindow()     {         InitializeComponent();     } }</pre>
---	--

Свойството `x:Class` в `xaml` тагът на `Window` трябва да съвпада с името на класа в `C#` за да може средата да обедини двете части на класа.

Това е и причината в `Code-Behind` файла класът да е дефиниран като `partial`.

**Пример:** Следните две проени на свойства в `Markup` и `Code-Behind` са еквивалентни:

<pre>&lt;Window x:Class="StudentInfoSystem.MainWindow" ... Title="Студентска информационна система" Height="350" Width="525"&gt;     &lt;Grid&gt;      &lt;/Grid&gt; &lt;/Window&gt;</pre>	<pre>public partial class MainWindow : Window {     public MainWindow()     {         InitializeComponent();         this.Title = "Студентска информационна система";     } }</pre>
--	---



### 3.3.1. Видове класове за прозорци

В `WPF` има 3 вида прозорци, представляващи 3 класа в `.Net`:

1. Класът `Window`:

Визуален интерфейс в прозорец, където всеки елемент (контрол) се намира вътре в прозореца. Прозорецът се визуализира самостоятелно. Цялото потребителско пространство е използваемо и подлежи на дефиниране със `XAML`.

2. Класът `NavigationWindow`:  
Специален вид прозорец, който наследява класа `Window`, но има навигационен панел в горната част (със стрелки за „Предишна“ и „Следваща“ страница). Т.е. приложение с множество последователни екрани, например, може да се възползва от таква функционалност.
3. Класът `Page`:  
Подобен на класа `Window`, но съдържанието му се визуализира в `NavigationWindow` или `Frame` (или браузър като ХВАР приложение), т.е. не се визуализира самостоятелно.

### 3.4. Файлът `App.xaml`

`App.xaml` е стартовата точка на приложението

- В `App.xaml` е деклариран клас наследяващ класа `Application`
- Подобно на `Window`, класът наследяващ `Application` е деклариран в два файла – Markup и Code-Behind

**Пример:** Двете декларации на класа `Application`

<pre>&lt;Application x:Class="StudentInfoSystem.App" xmlns="http://schemas.microsoft.com/ winfx/2006/xaml/presentation" xmlns:x="http://schemas.microsoft.com/ winfx/2006/xaml" xmlns:local="clr-namespace: StudentInfoSystem" StartupUri="MainWindow.xaml"&gt; &lt;Application.Resources&gt;  &lt;/Application.Resources&gt; &lt;/Application&gt;</pre>	<pre>namespace StudentInfoSystem {     public partial class App : Application     {     } }</pre>
--	---

Свойството `StartupUri` указва кой прозорец да се зареди при старт на програмата.

```
StartupUri="MainWindow.xaml"
```

В нашия случай стартира `MainWindow.xaml`

## 4. Основни контроли

### 4.1. Какво е контрола?

В WPF “контрола” е обединяващ термин, който се прилага за категория от WPF класове, които се ползват в прозорец или страница, имат графичен потребителски интерфейс и имплементират някакво поведение.

#### 4.1.1. Как се добавят контроли към прозорец?

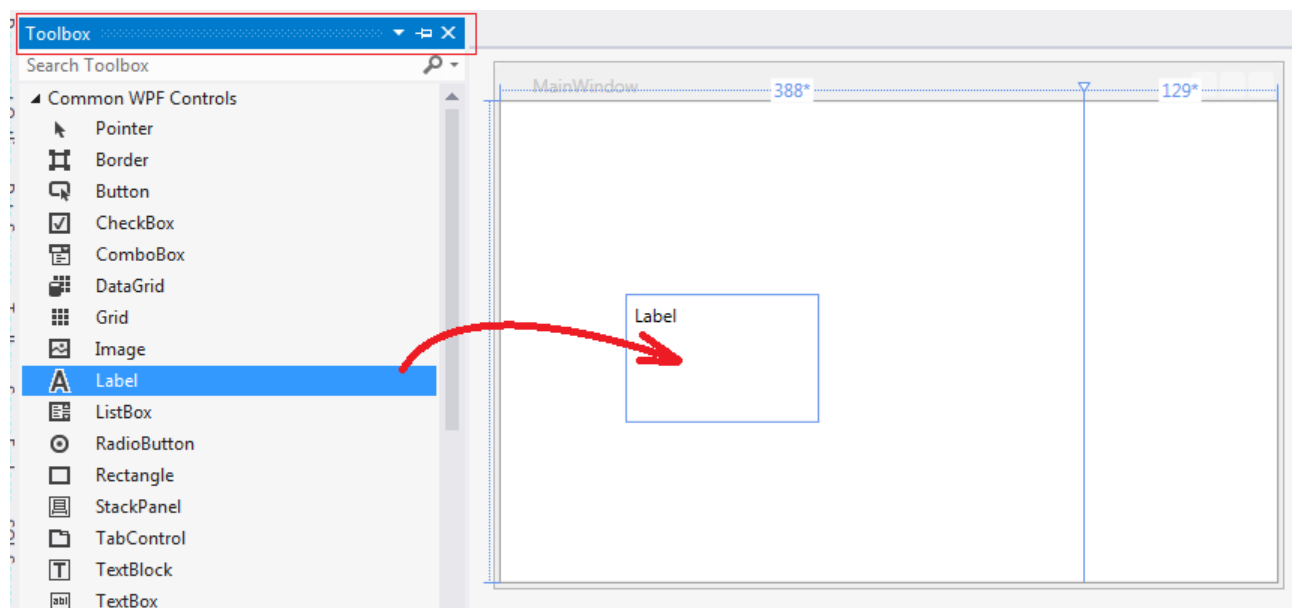
**Задача:** Добавяме нов проект към решението (solution-a):

- ("File" → "Add" → "New Project ...")  
(Или от "Solution Explorer" → "Solution" →  → "Add" → "New Project ...")
- В диалоговия прозорец изберете "Visual C#" → "Windows" → "WPF app"  
(За Visual Studio 2017: "Visual C#" → "Windows Classic Desktop" → "WPF app")
- Кръстете новия проект **WPFhello**.

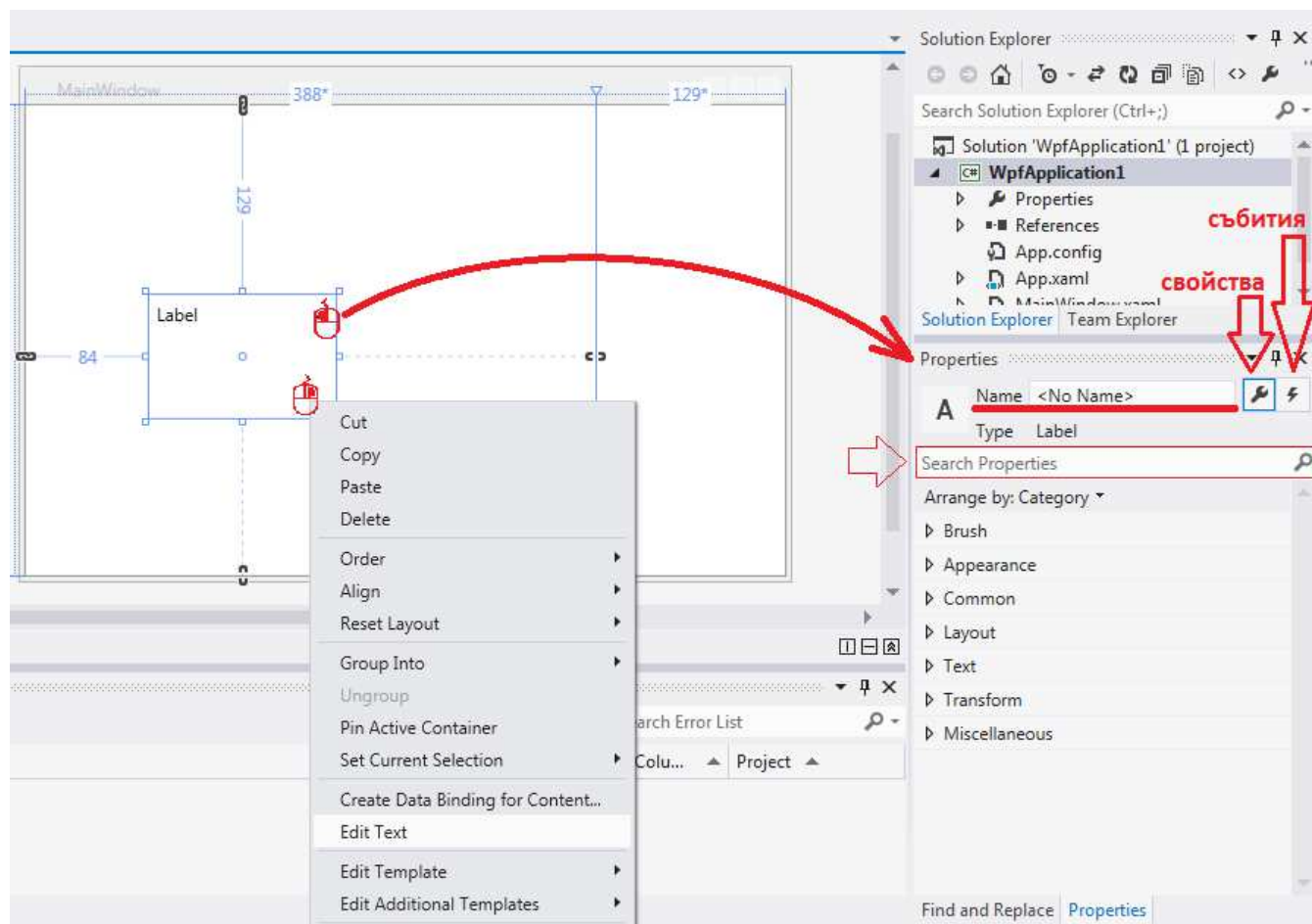
В проекта **WPFhello**:

**Задачи:**

За да добавим класическото съобщение „**Hello, WPF!**“ към нашия прозорец, ни е необходима контрола от типа **TextBlock** или **Label**.



Сменете името на контролата, текста, размера на шрифта и местоположението с помощта на панела със свойствата.



Стартирайте приложението (от менюто изберете Debug -> Start debugging или натиснете **F5**).  
Ако няма грешки, то честито първо приложение с WPF:



#### Задачи:

1. Добавете един бутон към прозореца на вашата програма. Кръстете го **btnHello** и променете текста му на „Здрасти!“.
- 1.1. Създайте негов **OnClick** метод  
(Може и като натиснете двойно върху бутона в дизайнера).
- 1.2. В новосъздадения метод добавете следния код:

`MessageBox.Show("Здрасти!!! Това е твоята първа програма на Visual Studio 2012!");`



Стартирайте програмата с **F5** или зеления триъгълник.

2. Добавете GroupBox контрола, която ще обгражда визуално всички контроли с въвеждане на данните за един потребител. Дайте ѝ име gbUser и променете текста ѝ.

2.1. Добавете Label и TextBox контроли.

2.2. Именувайте ги подходящо (например lblName и txtName).

2.3. Един GroupBox може да съдържа само една контрола като тяло, за това за сега ще го използваме само визуално да обгражда другите контроли в интерфейса.

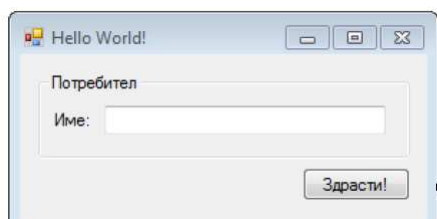
Може да следвате следния подход:

2.3.1. Поставете двете контроли и GroupBox контролата на едно ниво

2.3.2. Задавайте местоположение на Label и TextBox контролите

2.3.3. Задавайте размери на GroupBox контролата така че да обгражда другите контроли

Прозореца ви трябва да изглежда по подобен начин:



Стартирайте програмата с **F5** или зеления триъгълник.

3. Модифицирайте кода на бутона btnHello по следния начин:

```
MessageBox.Show("Здрасти " + txtName.Text + "!!! \nТова е твоята първа програма на Visual Studio 2012!");
```

Стартирайте програмата с **F5** или зеления триъгълник.

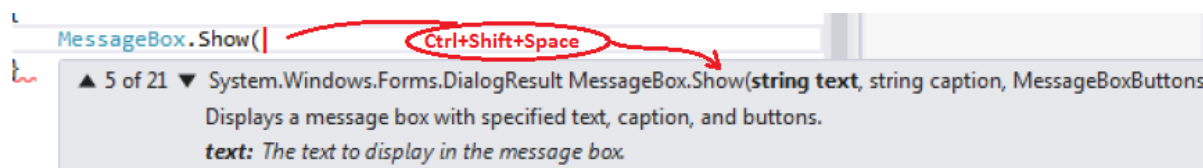
4. Направете проверка за дължината на текста, въведен от потребителя. Поставете ограничение за дължина на името поне 2 символа. Направете, така че само ако въведената информация отговаря на условията, да се извежда съобщението за поздрав, а в противен случай изведете съобщение с подкана за корекция. (Използвайте оператора "If-else").

5. Добавете необходимите компоненти и код за пресмятане на **n!** или **n<sup>y</sup>**, където n (и y) се въвежда от потребителя.

6. Добавете метод, който се изпълнява при изход от програмата и проверява дали потребителя е сигурен дали иска да я затвори.

#### Подсказки:

- Намерете правилното **събитие на формата**, което отговаря за затварянето на програмата. Формата трябва все още да не се е затворила.
- Разгледайте възможностите на MessageBox.Show метода. Той има **21 различни имплементации**, всяка с различен на брой и вид параметри. Освен текст можете да показвате различни икони, текстове и бутони.
- Методът MessageBox.Show НЕ Е от тип void, така че вижте какво ви връща за да разберете кой бутон е натиснал вашия потребител.



- За да предотвратите затварянето на формата трябва да прекратите изпълнението на събитието. За това служи параметъра е, който има свойство Cancel.

e.Cancel = true;

## 4.2. Динамично обхождане на елементи в контейнер

Ако ни се налага многократно да правим едни и същи операции с подобни контроли е добра идея да напишем оптимизиран код, работещ независимо колко на брой са те, а не да пишем повтарящи се аналогични редове за отделните контроли.

- Таблиците Grid се разглеждат като контейнери, съдържащи всички елементи в тях.
  - Можем да ги обходим с 1 цикъл и в зависимост от типа им да направим нещо с тях.
  - Children е колекцията от всички тези елементи.
1. Добавете още две TextBox контроли за имена. Нека поздавим всички с едно съобщение.
  2. Задайте има на Grid елемента (представляващ тялото на прозореца). Например MainGrid.
  3. Променете OnClick метода на бутона, така че да извлича въведеното от всички TextBox контроли. (Използвайте оператора "foreach", като следния код).

```
foreach (var item in MainGrid.Children)
{
    if (item is TextBox)
    {
        s = s + ((TextBox)item).Text;
        s = s + '\n';
    }
}
```

Стартирайте програмата с **F5** или зеления триъгълник.

4. Направете експеримент като добавите още TextBox контроли, че те ще се изпишат без да сте добавили нито един ред код

Стартирайте програмата с **F5** или зеления триъгълник.

## 4.3. Настройка на свойства

1. Добавете контрола от типа **TextBlock** през дизайнера и вижте какъв XAML се генерира.

```
<TextBlock HorizontalAlignment="Left" Margin="37,32,0,0" TextWrapping="Wrap"
Text="TextBlock" VerticalAlignment="Top"/>
```

2. Сменете името на контролата и местоположението с помощта на панела със свойствата и текста през дизайнера. Вижте как се е изменил XAML кода.

```
<TextBlock x:Name="textBlock1" HorizontalAlignment="Left" Margin="95,87,0,0"
TextWrapping="Wrap" Text="Hello, XAML!" VerticalAlignment="Top"/>
```

С това име ще е видим и в Code-Behind: **textBlock1**  
Например `TextBlock1.Text = "Some text";`

3. Да редактираме директно XAML кода и да видим как ще се отрази в дизайнера. Премахнете свойството `Margin` и добавете ново свойство:

```
FontSize="72"
```

4. Да добавим нова контрола изцяло със XAML. Добавете:

```
<TextBlock HorizontalAlignment="Center" VerticalAlignment="Center" FontFamily="Arial
Black">
    I edit XAML myself!
</TextBlock>
```

Стартирайте приложението (от менюто изберете `Debug -> Start debugging` или натиснете **F5**).

Може да забележите, че в елемента `TextBlock` използвахме различни атрибути (свойства), за да центрираме текста в прозореца (`HorizontalAlignment` и `VerticalAlignment`) и `FontSize`, за да уголемим текста. Пълния списък със свойства за всеки елемент можем да видим в `Properties` прозореца.

## 4.4. Събития в XAML

Поведението на приложението се имплементира посредством функционалността, която отговаря на потребителското взаимодействие с програмата. Това включва обработката на събития (например натискане на меню, лента с инструменти или бутон) или обръщение към слоя на бизнес логиката или данните в отговор. В WPF поведението е основно имплементирано с код, който се асоциира с маркър. Този тип код се нарича `code-behind`.

Нека добавим код за един бутон в нашия XAML. Добавете следния код в `Grid`-а, след вече поставения `TextBlock`:

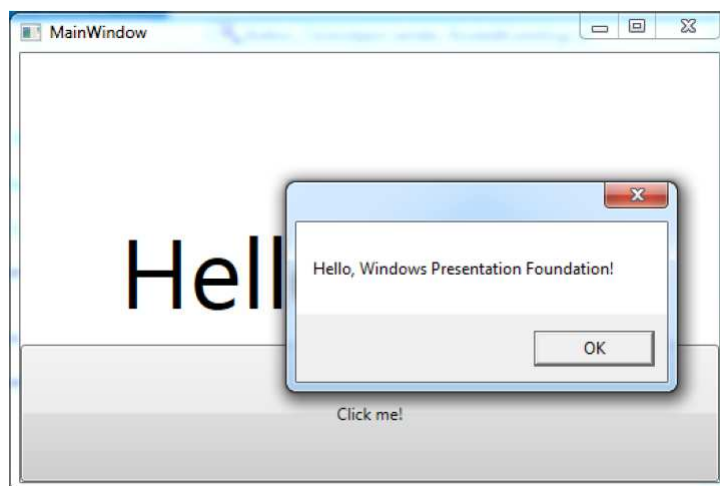
```
<Button HorizontalAlignment="Stretch" VerticalAlignment="Bottom" Height="100">
    Click me!
</Button>
```

В нашия пример, за да направим бутона да прави нещо, трябва да напишем неговия `code-behind` и да го свържем с маркър. Най-лесно това може да стане като изберете бутона и в

Properties прозореца на Visual Studio изберете да ви се покажат неговите събития. Изберете Click и щракнете два пъти върху него. Ще ви се генерира метода Button\_Click, а в XAML маркъпа ще се появи следния атрибут `Click="Button_Click"`. Добавете вече познатия MessageBox за извеждане на съобщение в Windows.

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    MessageBox.Show("Hello, Windows Presentation Foundation!");
}
```

Тествайте програмата с F5.



В този пример code-behind имплементира клас, който наследява Window класа. Атрибута **x:Class** се използва, за да свърже маркъпа с класа от code-behind файла. Методът **InitializeComponent** се извиква от конструктора на code-behind класа, за да свърже дефинирания с маркъп UI и code-behind класа. **InitializeComponent** се генерира автоматично при компилирането на програмата и не е нужно да имплементирате нищо ръчно. Комбинацията от **x:Class** и **InitializeComponent** осигурява на вашата програма правилна инициализация при създаването ѝ. Code-behind класа също имплементира event handler за click събитието на бутона. Така, когато се натисне бутона event handler-а показва съобщение със стандартния за Windows MessageBox.Show() метод.

Свойствата на контролите могат не само да се инстанцират и четат, но да се променят при изпълнение на програмата в Code-Behind.

**Пример:** Да променим съдържанието на `textBlock1` в зависимост от това кога е натиснат бутона:

Т.е. трябва да зададем стойност на свойството Text:

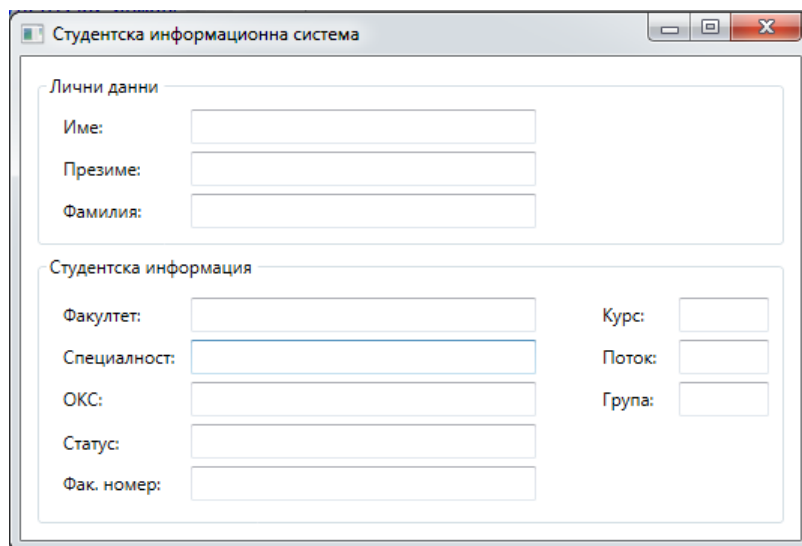
```
private void Button_Click(object sender, RoutedEventArgs e)
{
    MessageBox.Show("Hello, Windows Presentation Foundation!");

    textBlock1.Text = DateTime.Now.ToString();
}
```

## Общи задачи (пример за работа в час):

### В проекта StudentInfoSystem:

**Задача:** Добавете необходимите контроли за въвеждане/извеждане на основната информация за студента.



*Ако ви затруднява, пропуснете GroupBox контролите.*

Класът MainForm може да бъде разширяван и в дефиницията в Code-Behind (не само чрез тагове в Markup):

**Задача:** Добавете член-функции в класа MainForm, които да реализират следните функционалности:

*MainForm е клас в нашия проект. Не сме задължени всички функции в него да са орбаотващи събития на бутони. Можем да си го разширим с наши функции като долните, които да викаме по-нататм когато ни е удобно.*

- Метода за изчистване на всички контроли на формата.  
(Да им премахва съдържанието.)
- Метода за извеждане на данните на студент в контролите на формата.  
(Класът Student се намира в StudentRepository)
- Метод, който прави всички контроли на формата неактивни.  
(Свойството isEnabled. Всяка контрола има това свойство. Свойството isEnabled е булево. Съответно isEnabled = false прави контролите неактивни.).

*(Идеята – в бъдеще да можем да забраним работа с контролите, напр. при неподходящ потребител (анонимен).)*

- Метод противоположен на горния метод – който активира всички контроли

*За тестови цели може да поставите временни бутони, които да викат горните функции.*

Запишете си проекта! USB, DropBox, GoogleDrive, e-mail, SmartPhone, където и да е.

До края на упражненията ще работите върху този проект.

Вкъщи също ще работите върху този проект.

Накрая трябва да го представите завършен.