

Программирование С#

7. Приложения Windows Forms

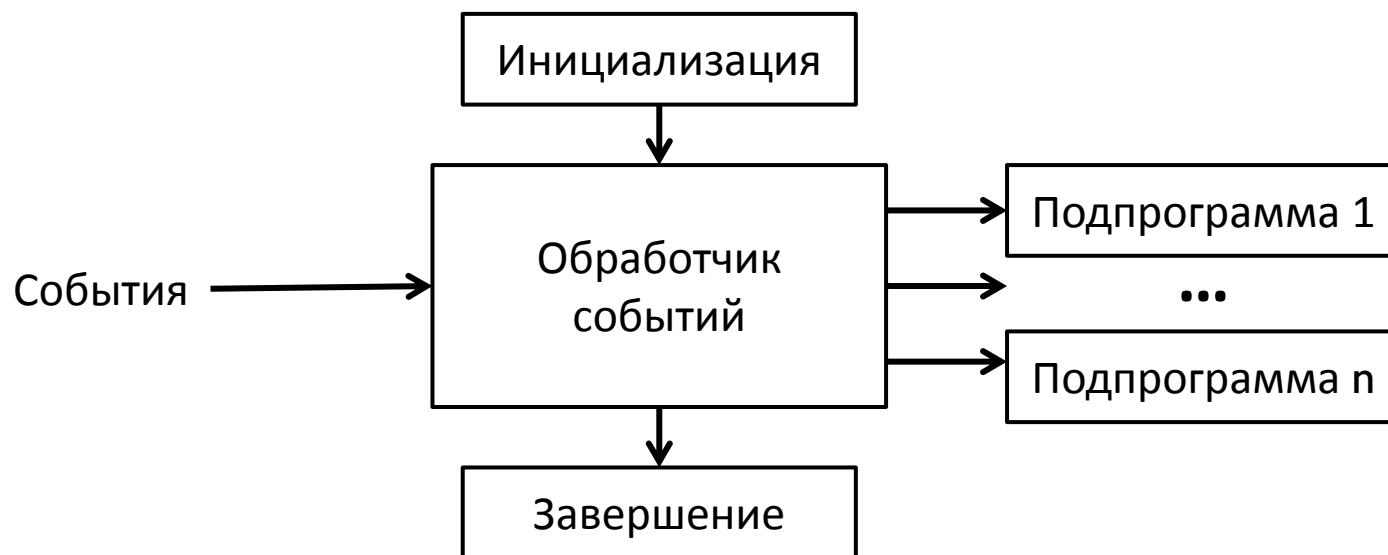
Карбаев Д.С., 2016

Особенности Windows

- ▶ Многозадачность — это возможность одновременно выполнять несколько приложений.
- ▶ Независимость программ от аппаратуры.
- ▶ Стандартный графический интерфейс с пользователем.
- ▶ Интерфейсные компоненты обращаются к аппаратуре не непосредственно, а через функции операционной системы, называемые API (Application Program Interface — программный интерфейс приложения). **API-функции** находятся в **динамических библиотеках** (Dynamic Link Library, **DLL**), разделяемых всеми приложениями.
- ▶ Поддержка виртуального адресного пространства для каждого приложения.
- ▶ Возможность обмена данными между приложениями.
- ▶ Возможность запуска старых программ.

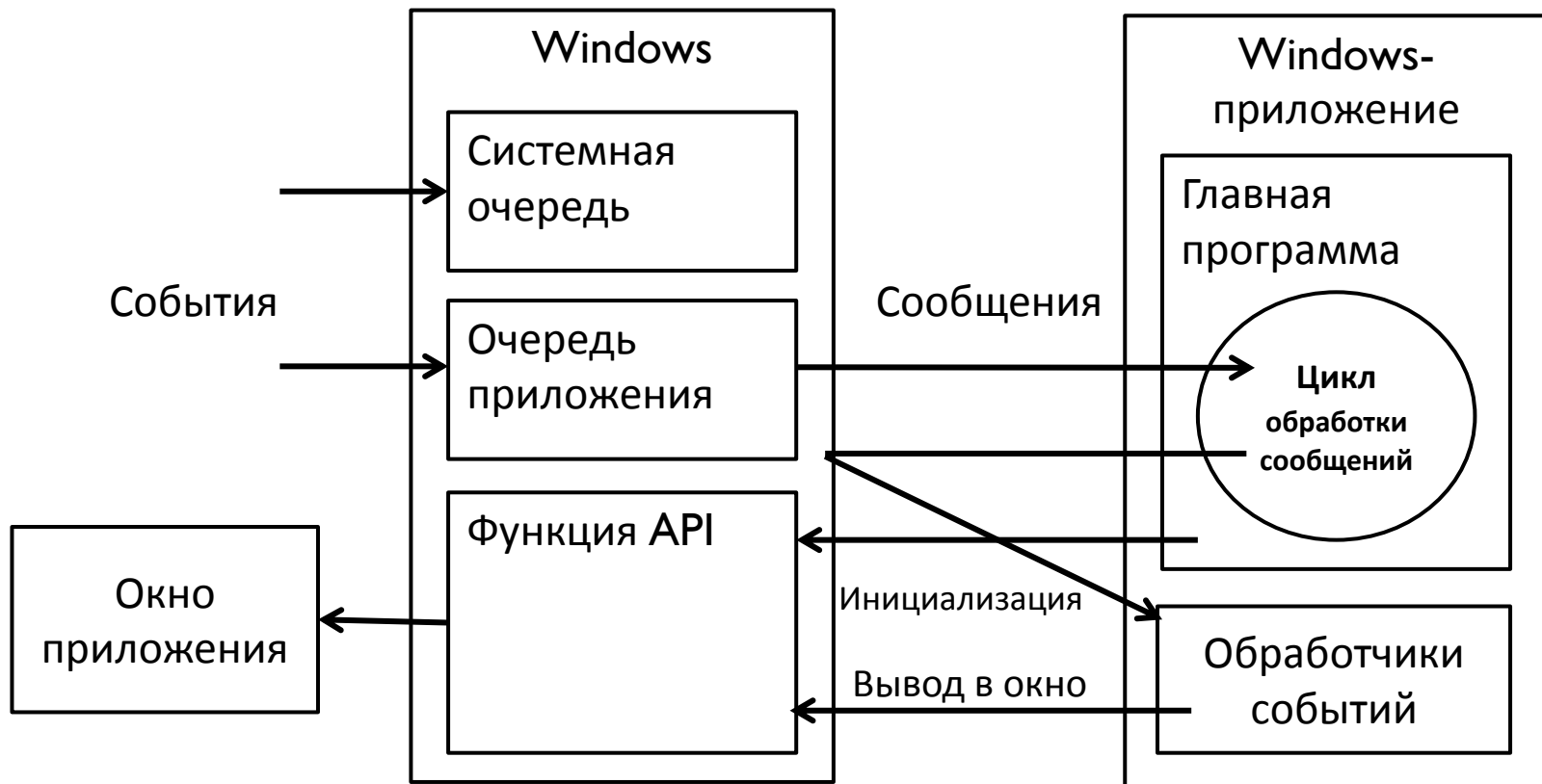
Событийное управление

- ▶ Событие воспринимается Windows и преобразуется в **сообщение**, т.е. запись, содержащую необходимую информацию о событии.
- ▶ Сообщения поступают в общую очередь, откуда распределяются по очередям приложений. Каждое приложение содержит **цикл обработки сообщений**, который выбирает сообщение из очереди и через операционную систему вызывает подпрограмму, предназначенную для его обработки.

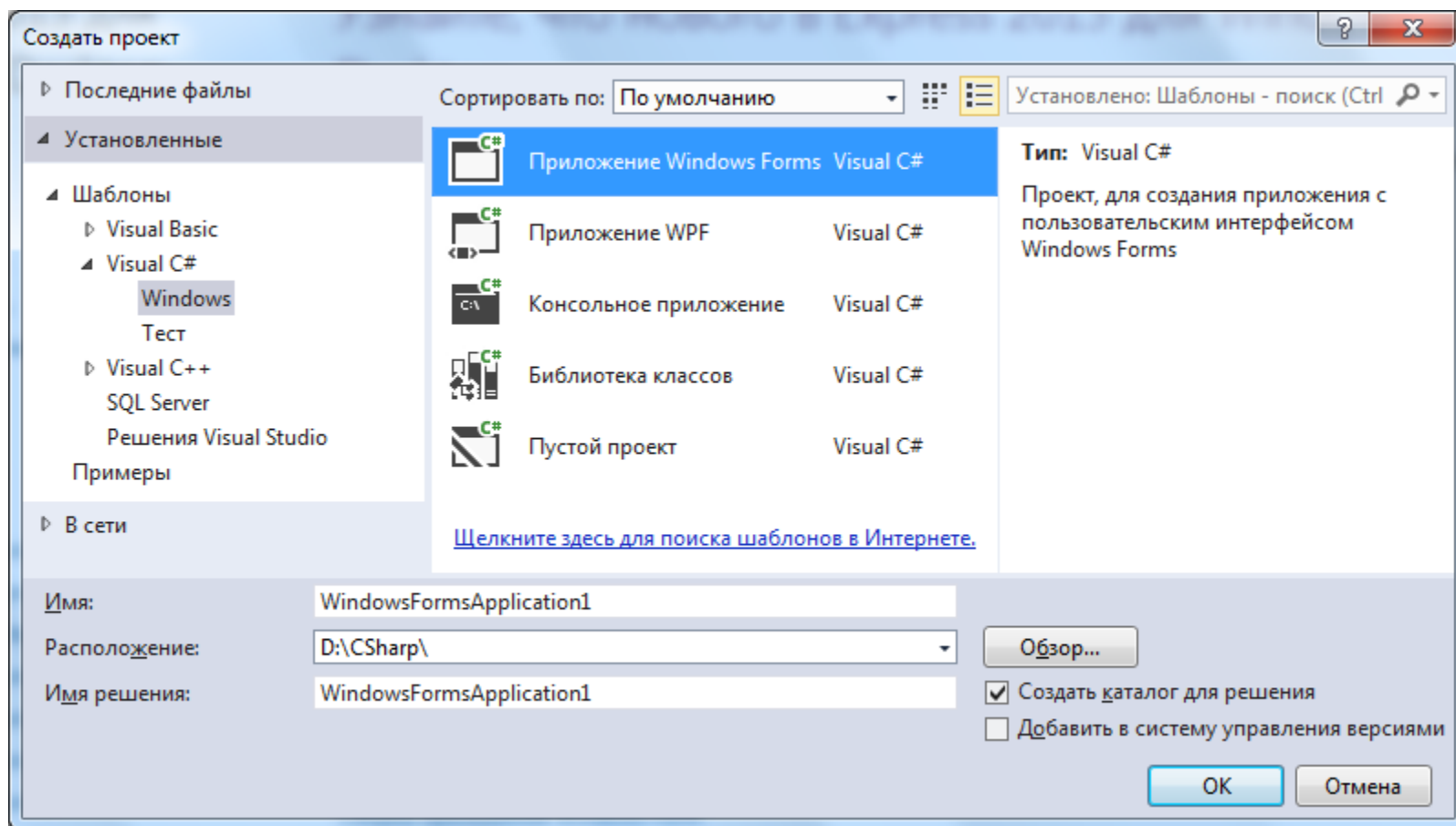


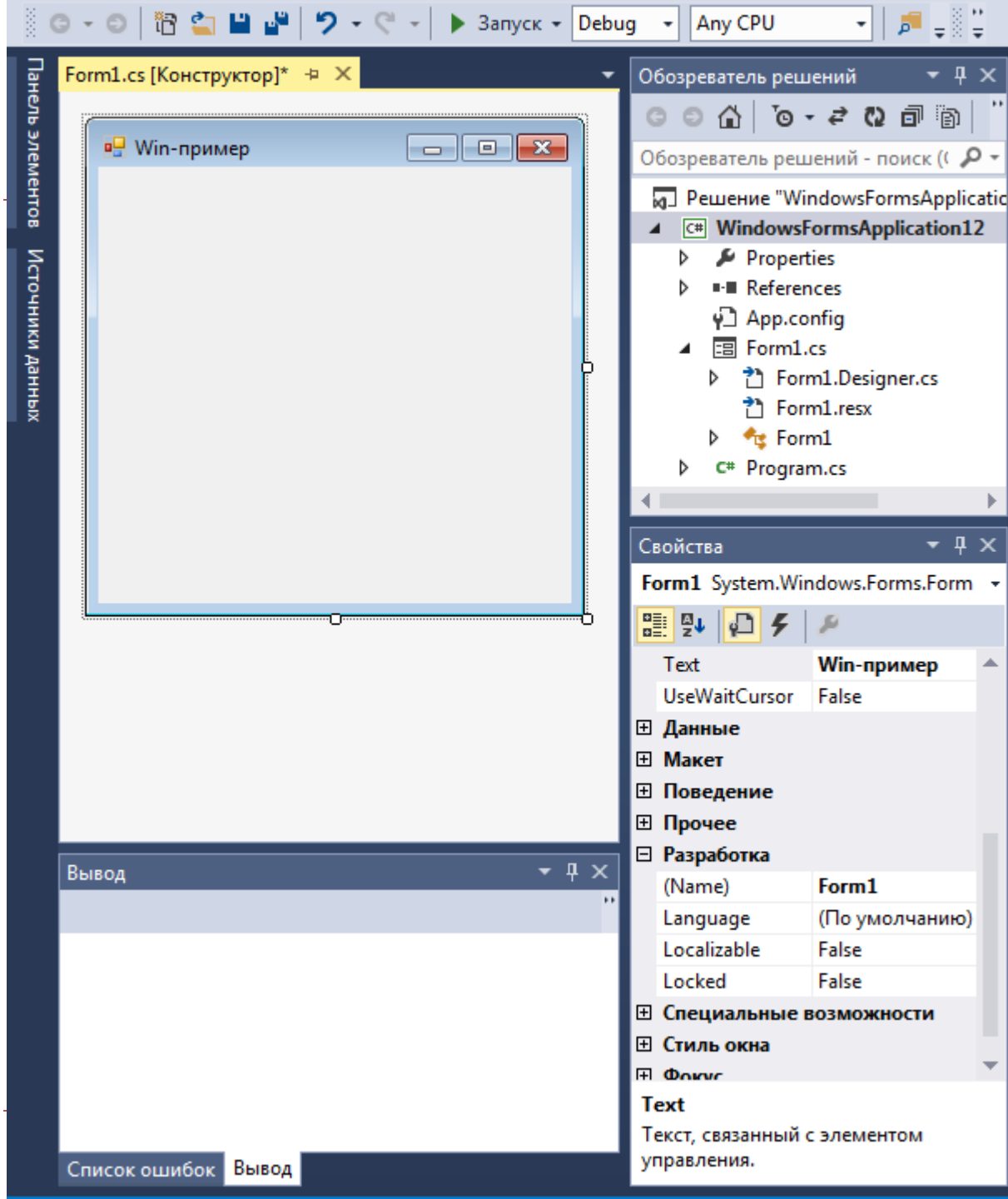
Событийное управление

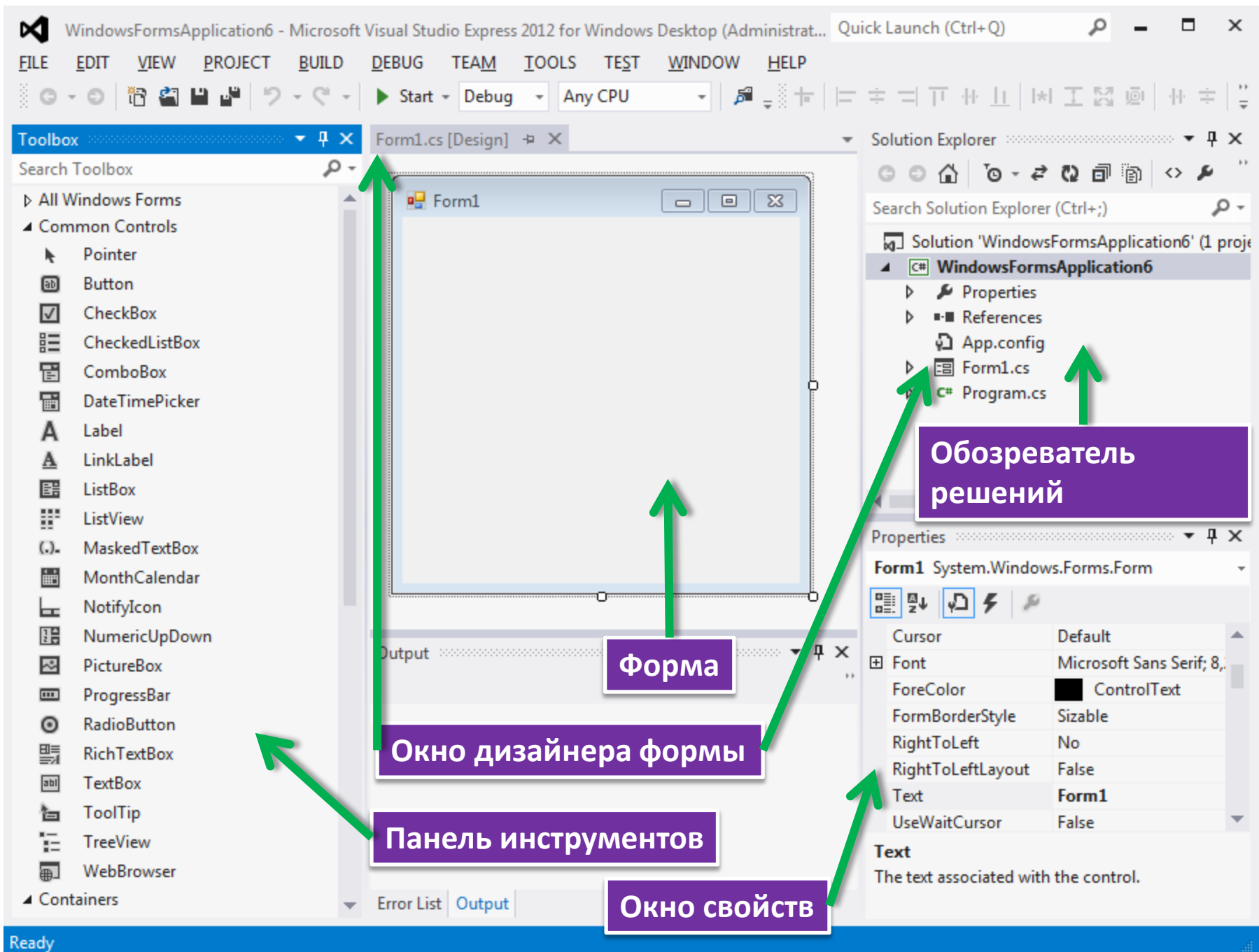
- Таким образом, Windows-приложение состоит из главной программы, обеспечивающей инициализацию и завершение приложения, цикла обработки сообщений и набора *обработчиков событий*.



Шаблон Windows-приложения





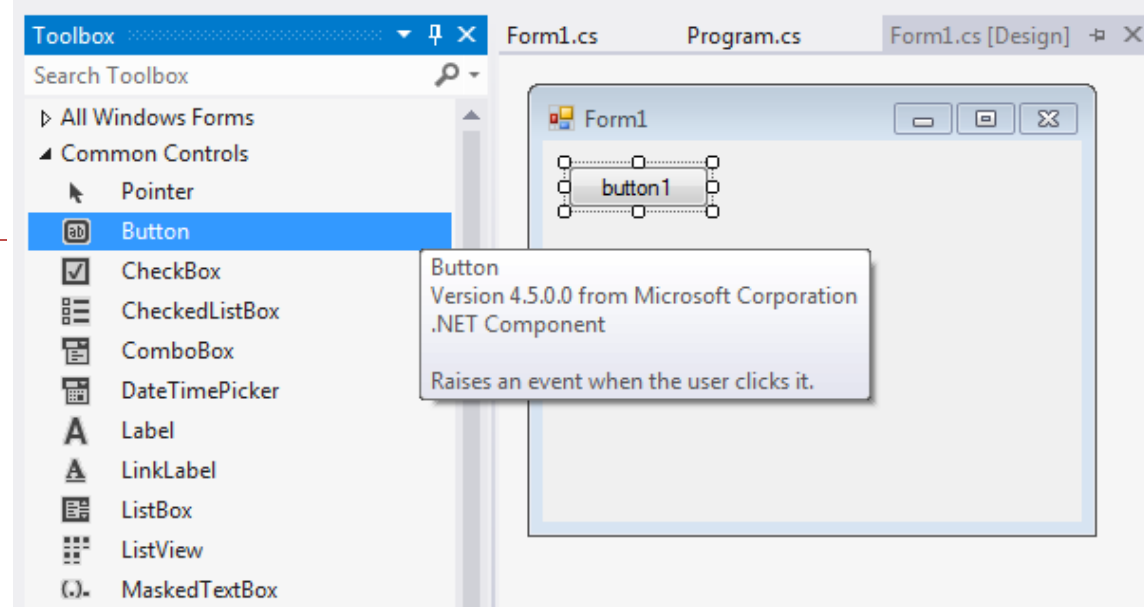


Быстрый старт

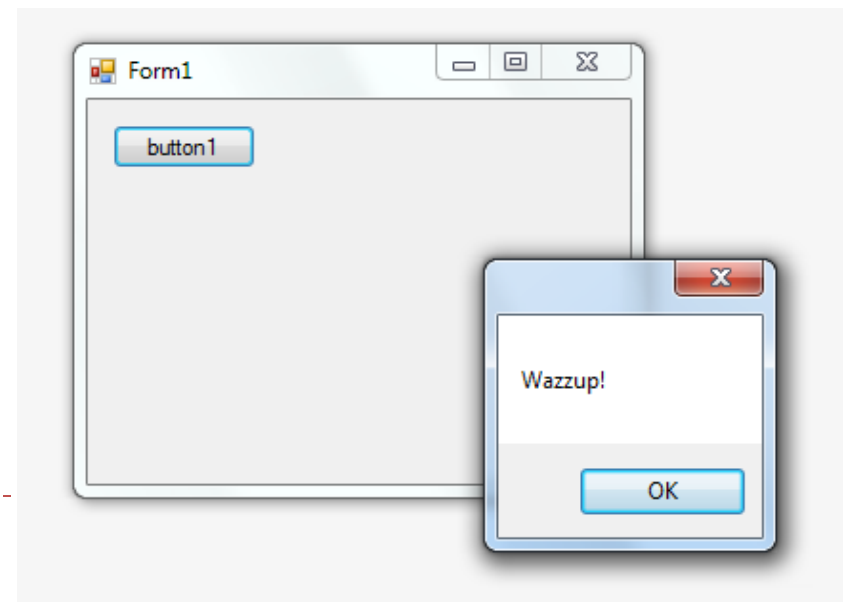
Создадим проект Windows Forms, перетащим на форму кнопку Button, двойной клик по ней.

Пропишем в методе >>>

Запустим проект и нажмем на кнопку.



```
private void button1_Click(object sender, EventArgs e)
{
    MessageBox.Show("Wazzup!");
}
```



Шаблон Windows-приложения

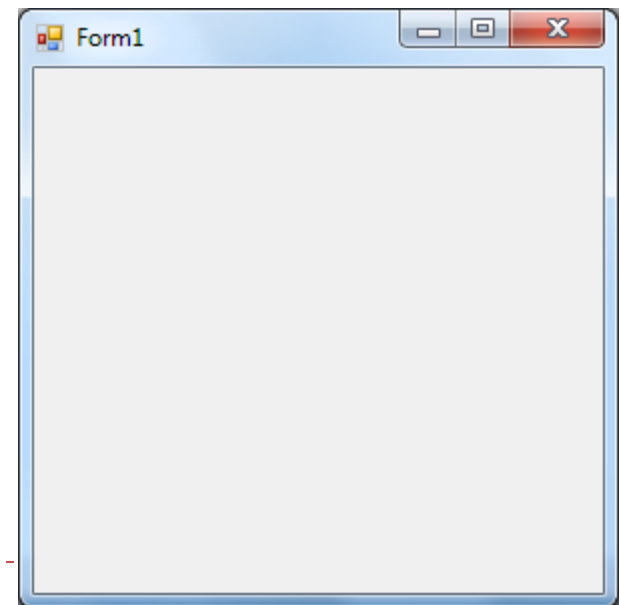
- ▶ Форма представляет собой окно и предназначена для размещения **компонентов** (контролов, виджетов, элементов управления, элементов интерфейса) — меню, текста, кнопок, списков, изображений и т. д. см. [Элемент интерфейса на Википедии](#).
- ▶ Среда создает не только заготовку формы, но и шаблон текста приложения. Перейти к нему можно, щелкнув в окне Solution Explorer: View - Solution Explorer (Вид – Обзорщик решений) правой кнопкой мыши на файле **Form1.cs** и выбрав в контекстном меню команду «View Code» (Просмотр кода).

```
namespace WindowsApplication1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
    }
}
```

Шаблон Windows-приложения

- ▶ В Win-приложении метод Main() находится в файле **Program.cs**; в Main() происходит вызов нашего класса Form1

```
static void Main() {  
    Application.EnableVisualStyles();  
    Application.SetCompatibleTextRenderingDefault(false);  
    Application.Run(new Form1());  
}
```



Свойства окна

0. **Name** – Задаёт имя объекта формы.

1. **FormBorderStyle** - Позволяет задавать рамку окна, а следовательно и его поведение, то есть фиксированный, растягиваемый и т.д.;

2. **Text** - Задаёт заголовок окна.

3. **MaximumSize** и **MinimumSize** - задают соответственно максимальный и минимальный размер окна.

4. **StartPosition** - Задаёт позицию в которой будет первоначально формироваться окно.
Например: CenterScreen или CenterParent.

5. **WindowState** - позволяет задать состояние окна (свернутое или развёрнутое на весь экран).

6. **Icon** - Задаёт иконку для окна.

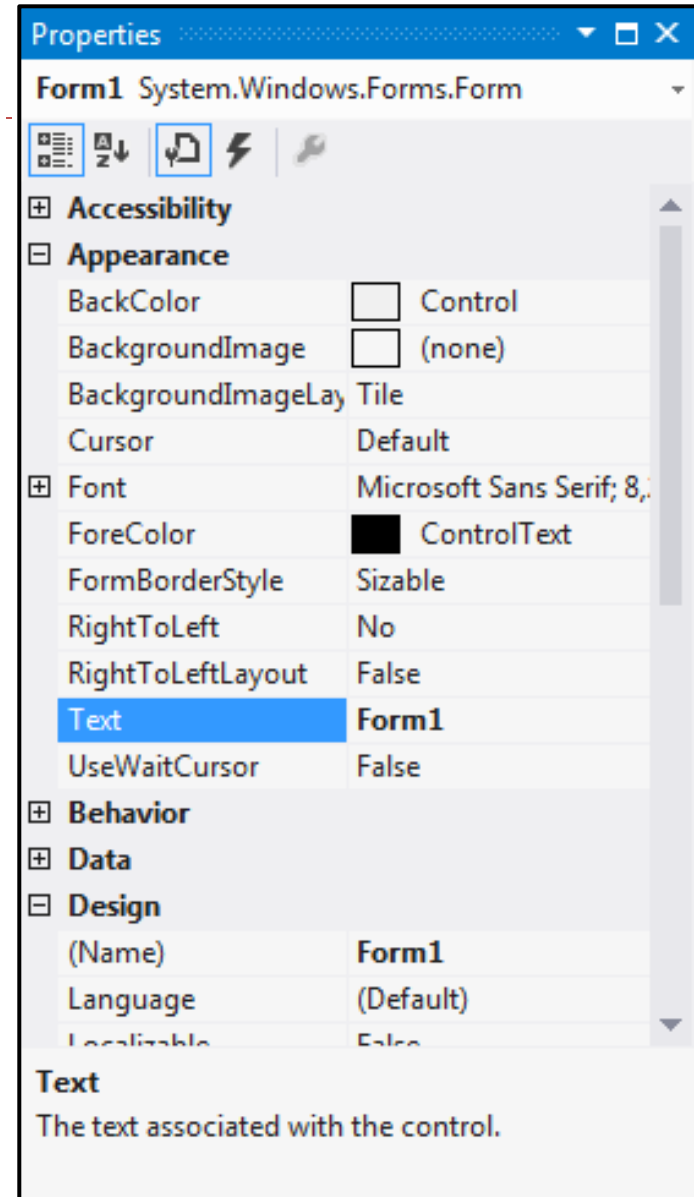
7. **MaximizeBox** и **MinimizeBox** - позволяет скрыть кнопки управления состоянием окна (кнопки в правом верхнем углу окна).

8. **Opacity** - задаёт прозрачность окна.

9. **ShowIcon** - Показывать или нет иконку.

10. **ShowInTaskBar** - Показывать или нет окно в Windows Explorer.

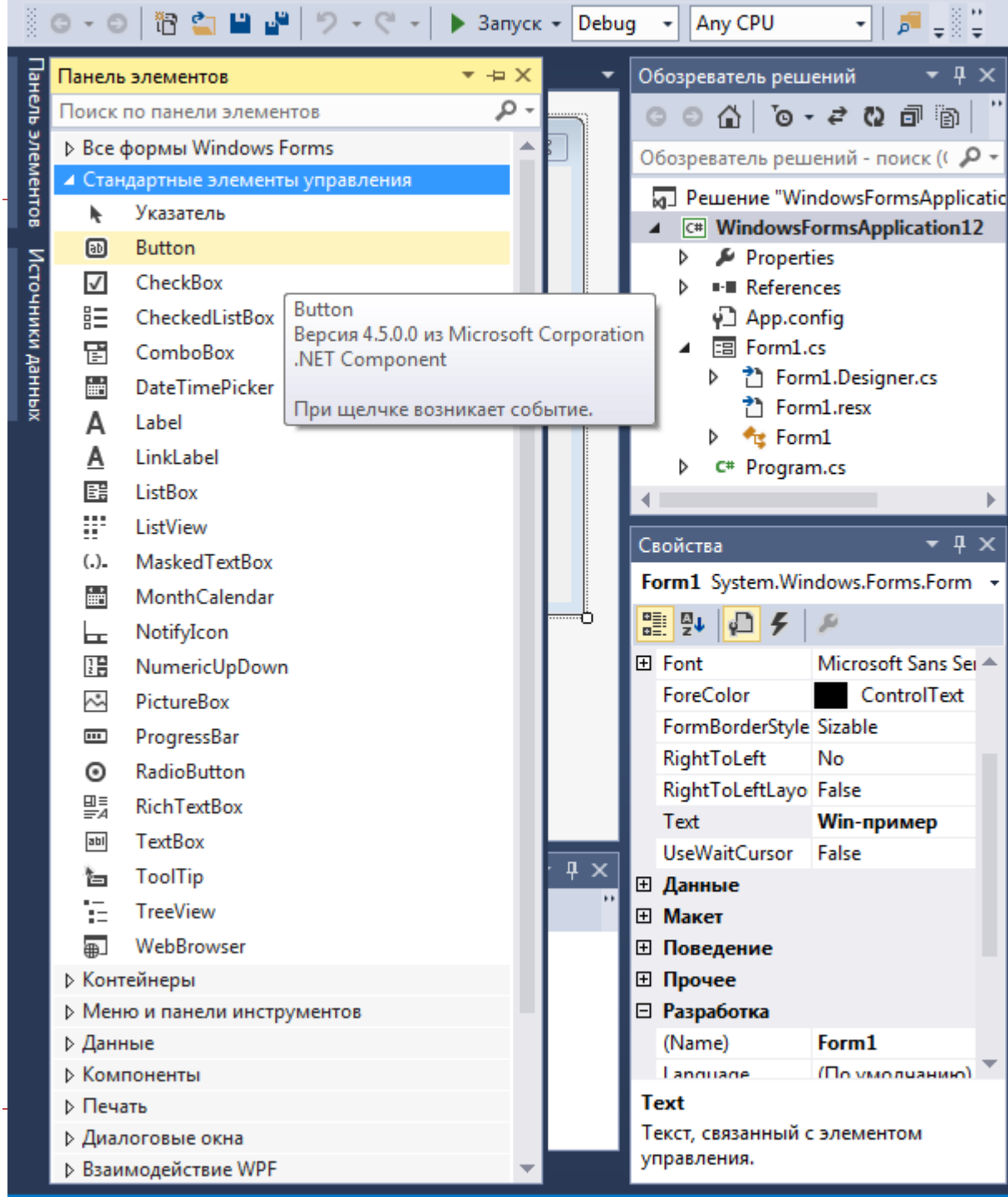
11. **TopMost** - Окно будет по верх остальных окон.



Элементы интерфейса

Существует стандартный набор элементов интерфейса, включающий следующие элементы управления:

- ▶ [кнопка](#) (*button*)
- ▶ [радиокнопка](#) (*radio button*)
- ▶ [флажок](#) (*check box*)
- ▶ [значок](#) (иконка, *icon*)
- ▶ [список](#) (*list box*)
- ▶ дерево — [иерархический список](#) (*tree view*)
- ▶ [раскрывающийся список](#) (*combo box*, *drop-down list*)
- ▶ поле редактирования (*textbox*, *edit field*)
- ▶ элемент для отображения табличных данных (*grid view*)
- ▶ [меню](#) (*menu*)
 - ▶ главное меню окна (*main menu*)
 - ▶ [контекстное меню](#) (*popup menu*)
 - ▶ ниспадающее меню (*pull down menu*)
- ▶ [окно](#) (*window*)
 - ▶ [диалоговое окно](#) (*dialog box*)
 - ▶ [модальное окно](#) (*modal window*)
- ▶ панель (*panel*)
- ▶ [вкладка](#) (*tab*)
- ▶ [панель инструментов](#) (*toolbar*)
- ▶ полоса прокрутки (*scrollbar*)
- ▶ ползунок (*slider*)
- ▶ строка состояния (*status bar*)
- ▶ всплывающая [подсказка](#) (*tooltip*, *hint*)



Toolbox

Search Toolbox

▸ All Windows Forms

▾ Common Controls

- Pointer
- Button
- CheckBox
- CheckedListBox
- ComboBox
- DateTimePicker
- Label
- LinkLabel
- ListBox
- ListView
- MaskedTextBox
- MonthCalendar
- NotifyIcon
- NumericUpDown
- PictureBox
- ProgressBar
- RadioButton
- RichTextBox
- TextBox
- ToolTip
- TreeView
- WebBrowser

Кнопка

Флажок/галочка

Список флажков

Выпадающий список

Выбор даты/времени

Надпись

Ссылка

Список

Список ListView

Текстовое поле/поле ввода

Дерево выбора

Свойства кнопки Button

0. **Name** – имя объекта кнопки

1. **Text** - текст на кнопке

2. **FlatStyle** - определяет стиль кнопки

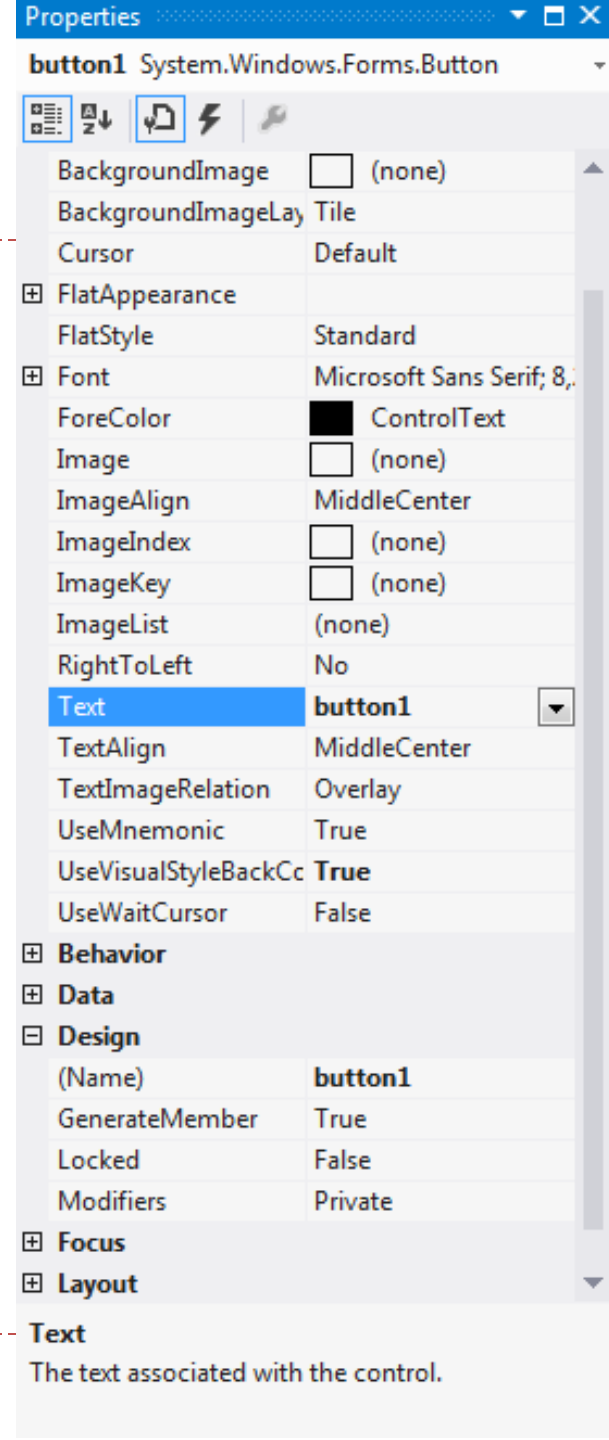
3. **Enabled** - в true кнопка доступна для нажатия

4. **Visible** - в false кнопка не отображается на форме, хотя она есть

5. **TabIndex** - определяет последовательность перехода на контрол по Tab-у

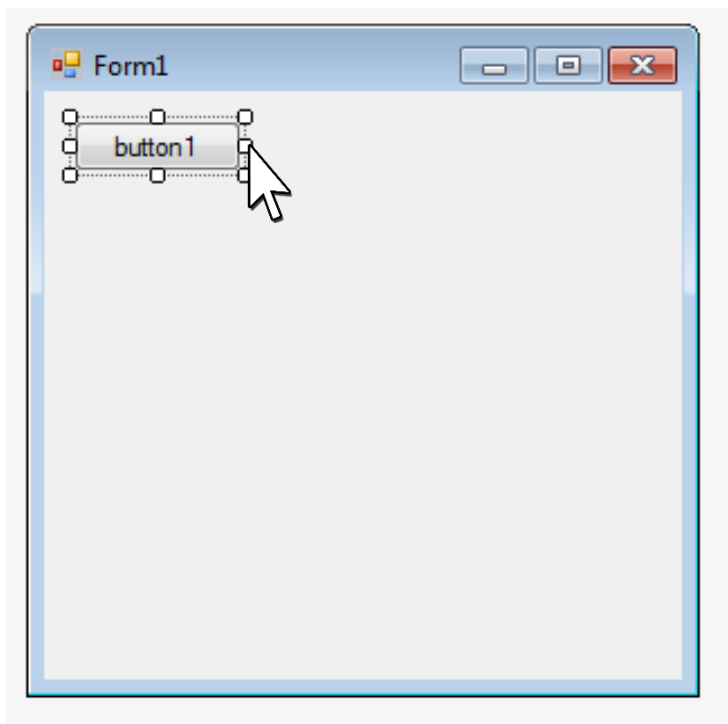
6. **Dock** - определяет заполнение одного из краев (или всех сразу) которое будет заполнять контрол, того контейнера в котором находится

7. **Anchor** - Определят какого края контейнера будет придерживаться контрол при растягивании формы



События кнопки Button

- ▶ По умолчанию срабатывает событие **Click**. Чтобы быстро вызвать обработчик данного события необходимо дважды кликнуть по кнопке в окне дизайнера.



```
//Файл Form1.cs
namespace WindowsFormsApplication6
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();

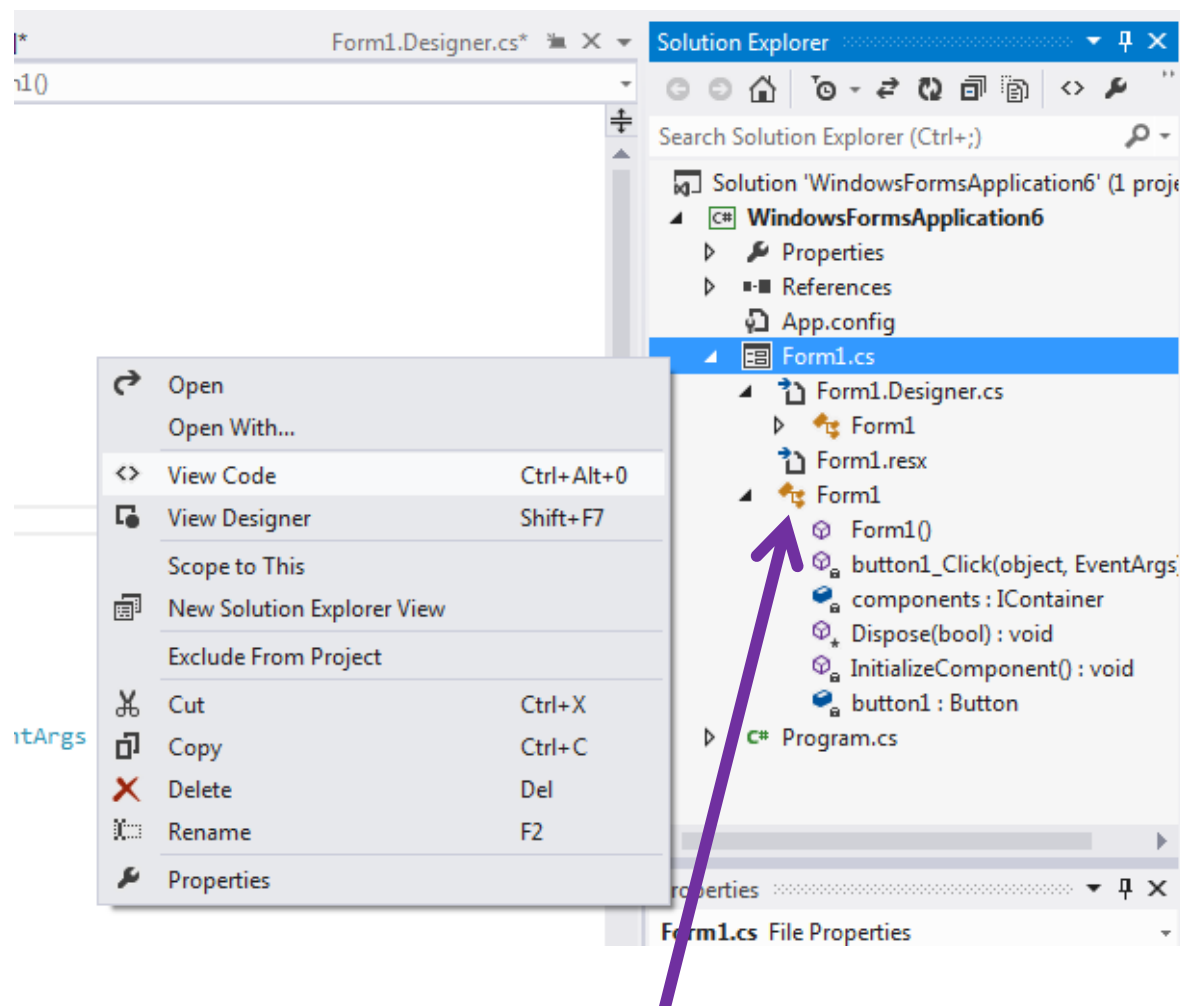
            private void button1_Click(object sender, EventArgs e)
            {

            }

        }
    }
}
```


Код событий формы

- ▶ Получить доступ к исходному коду событий формы можно кликнув правой кнопкой по файлу **Form1.cs** в Обозревателе решений, и выбрав пункт меню **ViewCode** (Просмотр кода)
- ▶ Можно посмотреть список полей и методов формы



Список полей и методов формы

Код дизайна формы (файл Form1.Designer.cs)

Генерируется
автоматически, не
изменять!



О, нет!
Они опять залезли в код
дизайнера!

```
WindowsFormsApplication6
Form1.Designer.cs*
WindowsFormsApplication6.Form1
components
namespace WindowsFormsApplication6
{
    partial class Form1
    {
        /// <summary> ...
        private System.ComponentModel.IContainer components = null;

        /// <summary> ...
        protected override void Dispose(bool disposing) ...

        #region Windows Form Designer generated code

        /// <summary> ...
        private void InitializeComponent()
        {
            this.button1 = new System.Windows.Forms.Button();
            this.SuspendLayout();
            //
            // button1
            //
            this.button1.Location = new System.Drawing.Point(13, 13);
            this.button1.Name = "button1";
            this.button1.Size = new System.Drawing.Size(75, 23);
            this.button1.TabIndex = 0;
            this.button1.Text = "button1";
            this.button1.UseVisualStyleBackColor = true;
            this.button1.Click += new System.EventHandler(this.button1_Click);
            //
            // Form1
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(284, 262);
            this.Controls.Add(this.button1);
            this.Name = "Form1";
            this.Text = "Form1";
            this.ResumeLayout(false);

        }

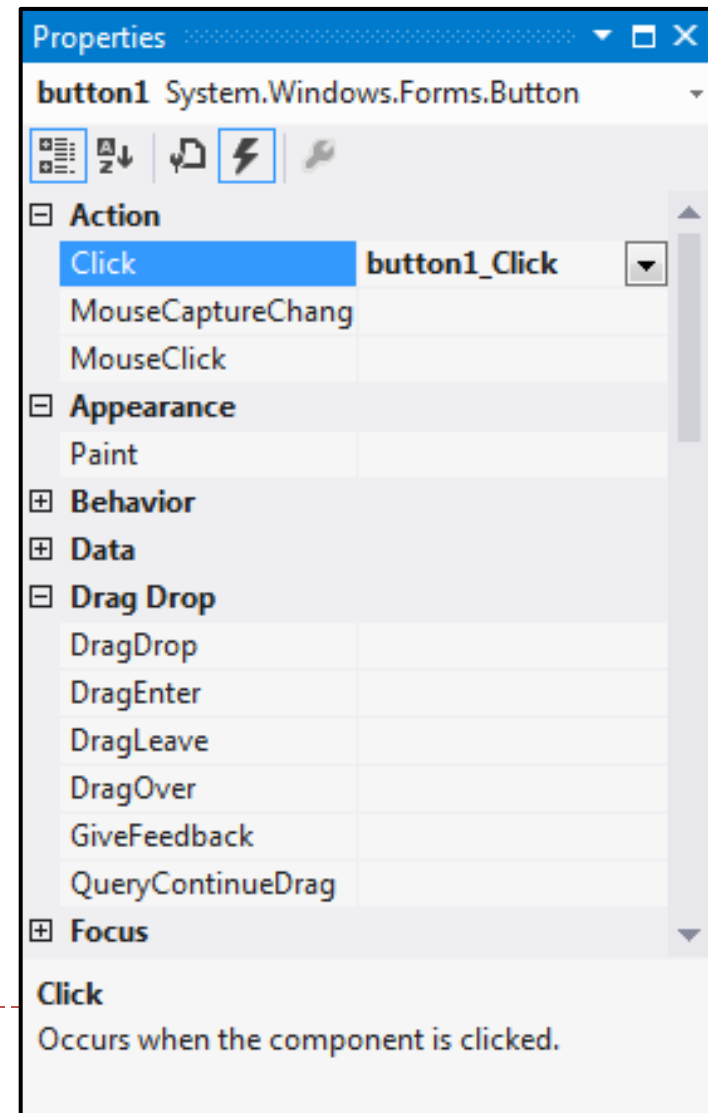
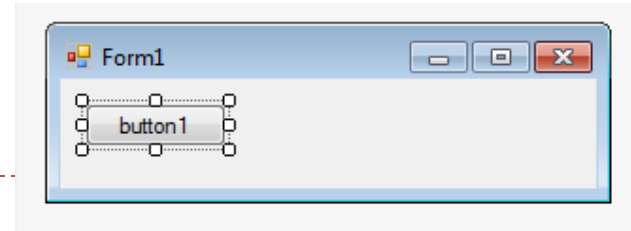
        #endregion

        private System.Windows.Forms.Button button1;
    }
}
```

События на форме

У каждого элемента на форме существует набор **событий** (клик, двойной клик, наведение курсором, нажатие клавиши и т.д.), для которых мы можем создать **метод-обработчик** (в нашем случае – `button1_Click`), который отобразится в свойствах элемента (значок молнии).

Чтобы переключиться обратно к свойствам формы, необходимо кликнуть на значок слева от молнии.

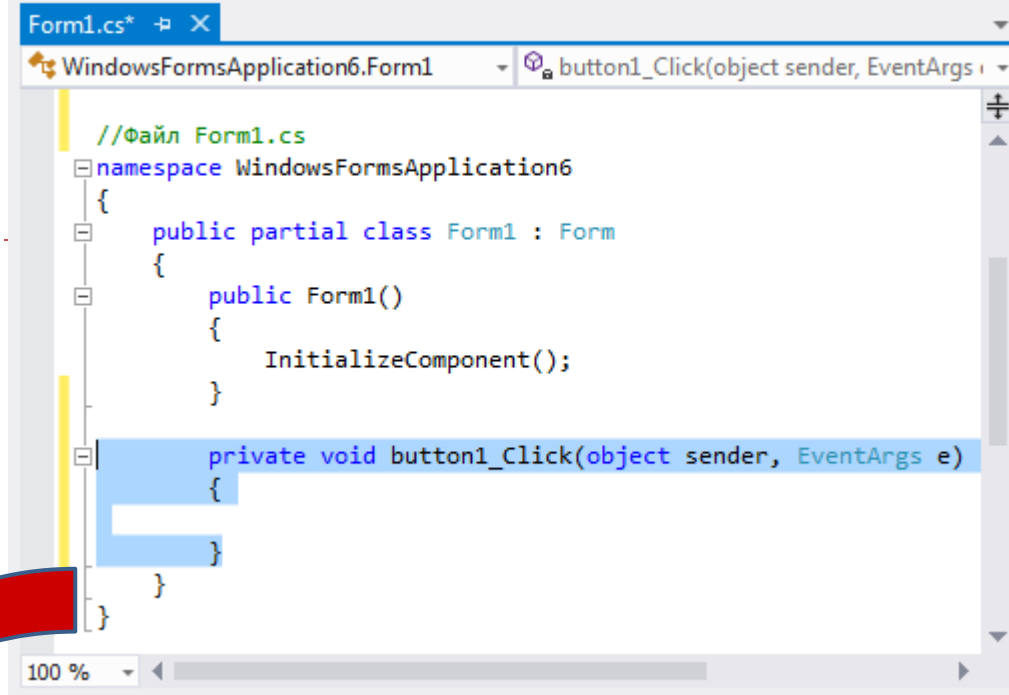


Ошибки на форме

Удалим вручную из файла Form1.cs созданный метод-обработчик нажатия на кнопку Button1...

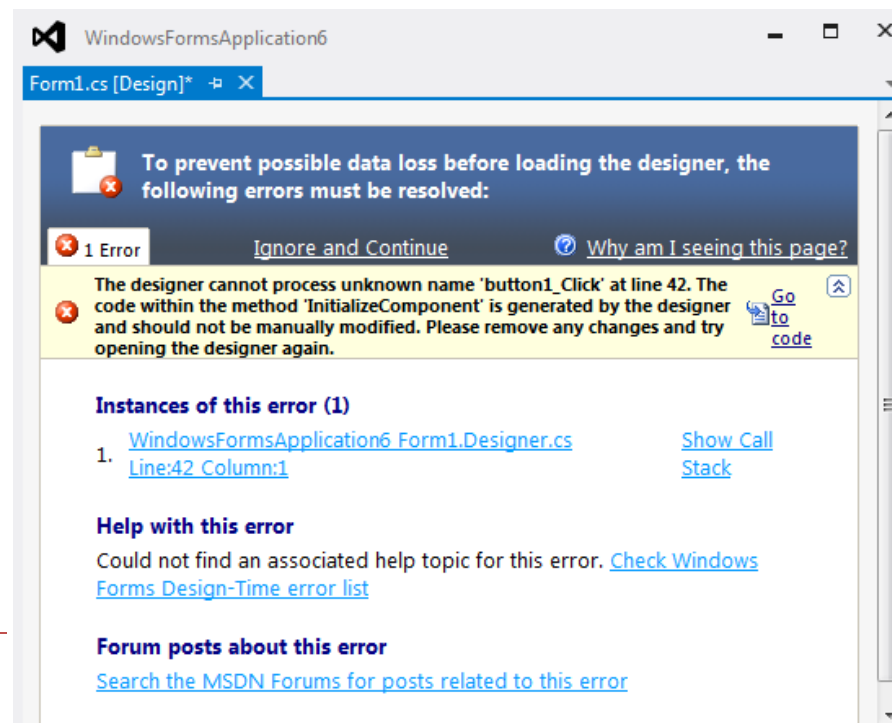
Упс....

И это далеко не самое страшное, что может случиться...



```
//Файл Form1.cs
namespace WindowsFormsApplication6
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
        }
    }
}
```



События кнопки Button

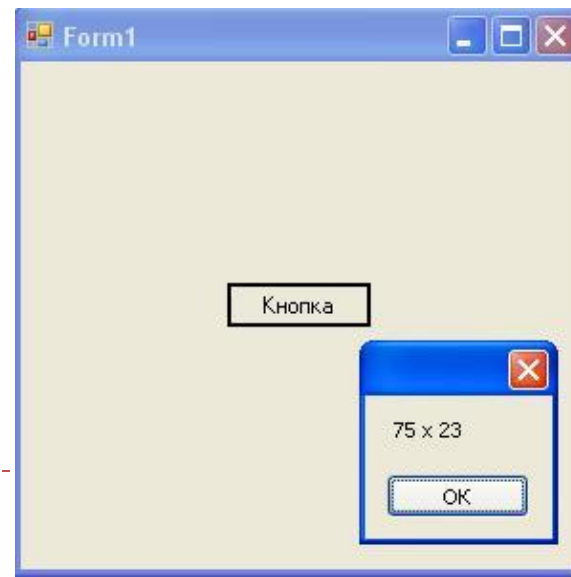
При создании обработчика `button1_Click()`, Visual Studio **автоматически** вставляет данную строку в файл-дизайнер (`Form1.Designer.cs`):

```
this.button1.Click += new System.EventHandler(this.button1_Click);
```

В случае, если метод `button1_Click()` отсутствует, можно удалить эту строку, и ошибка (см. пред. слайд) на форме исчезнет.

- ▶ В обработчике вы можете, например, показать размер кнопки

```
private void button1_Click(object sender, EventArgs e){  
    MessageBox.Show(button1.Size.Width + " x " + button1.Size.Height);  
}
```



Поле ввода TextBox

- ▶ Текстовое поле служит для ввода текста, иногда может использовать только для отображений.

Свойства:

0. **Name** – имя объекта поля ввода;

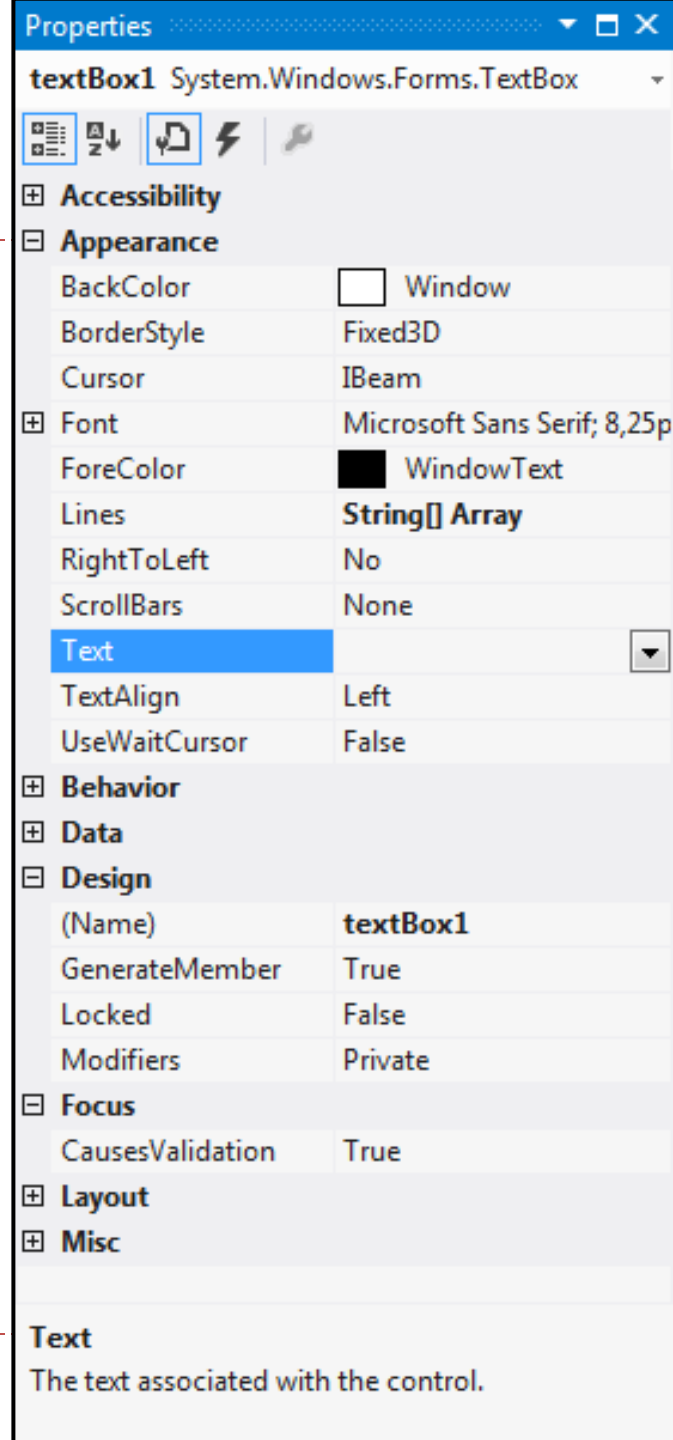
1. **Text** – собственно, строка введенного текста ;

2. **CharacterCasing** - возможность вводить только большие буквы или только строчные;

3. **MultiLine** - текстовое поле трансформируется в поле для ввода нескольких строк;

4. **PasswordChar** - если установлен какой либо символ, то поле маскирует ввод текста под пароль и текст скрывается под введенный символ (например звездочку);

5. **ReadOnly** - в true текстовое поле недоступно для редактирования.



События TextBox

- ▶ По умолчанию используется событие **TextChanged!!!**

Т.е. при двойном клике в дизайнере на `textBox1` создается событие `textBox1_TextChanged ()`.

- ▶ Добавляем на форму еще Label. Будем выводить то что сейчас есть в нашем текстовом поле :

```
private void textBox1_TextChanged(object sender, EventArgs e)
{
    label1.Text = textBox1.Text;
}
```



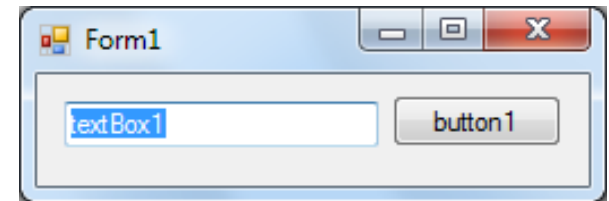
Не кликайте 2 раза по текстбоксам!



Переименование элементов формы

- ▶ Создадим форму с элементами: **textBox1** и **button1**.
- ▶ По нажатию на кнопку создадим простой метод:

```
private void button1_Click(object sender, EventArgs e)
{
    textBox1.Text = "Джинн благодарит тебя!";
}
```



- ▶ Переименуем элементы в:

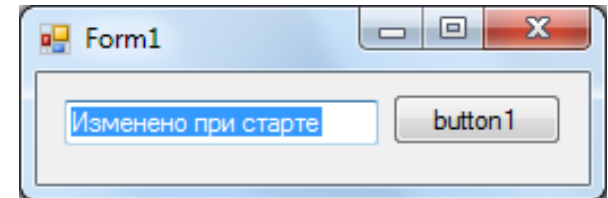
textBox1New и **button1New** (изменить свойство (Name) в разделе Design-Дизайн окна свойств). Visual Studio автоматически переименует все вхождения (результаты отобразятся в окне Output-Вывод).

```
private void button1_Click(object sender, EventArgs e)
{
    textBox1New.Text = "Джинн благодарит тебя!";
}
```

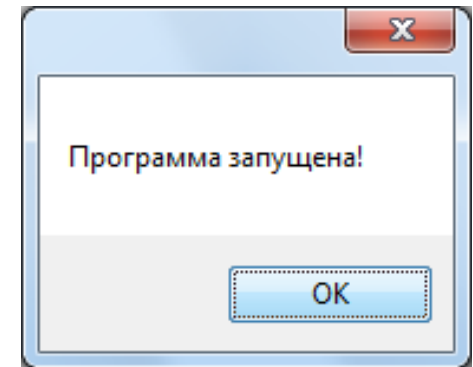
Обратите внимание, что изменилось название объекта текстового поля, а название метода осталось прежним, т.к. оно прописано в событии Click кнопки.

Инициализация элементов на форме

- ▶ Элементы форм можно настраивать в режиме дизайнера, а можно генерировать программно при запуске приложения (в конструкторе формы)



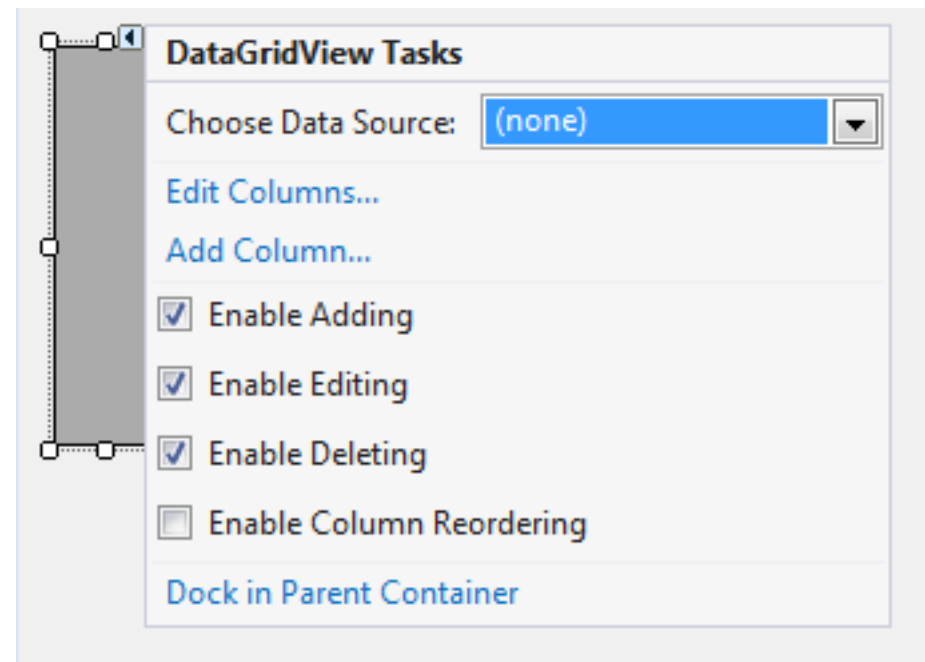
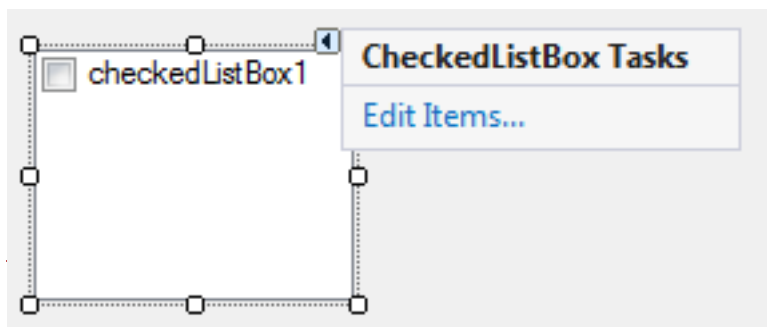
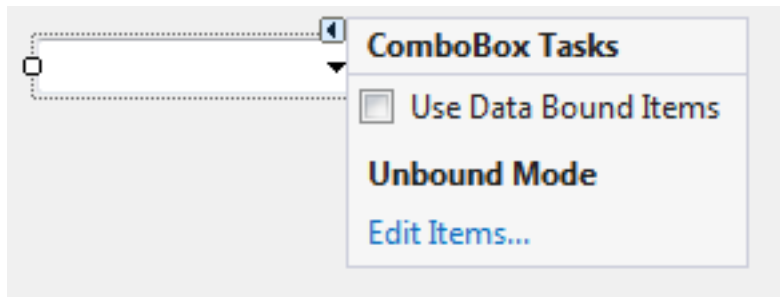
```
public Form1()  
{  
    InitializeComponent();  
    textBox1New.Text = "Изменено при старте";  
    MessageBox.Show("Программа запущена!");  
}
```



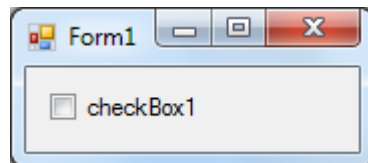
- ▶ Такую инициализацию удобно использовать для заполнения списков и таблиц.

Детальные настройки контролов

- ▶ Можно устанавливать дополнительные свойства контролов и вводить начальные значения в режиме дизайнера.



CheckBox



- ▶ CheckBox необходим для ответа ДА/НЕТ, хотя данный контрол имеет еще одно состояние - неопределенное.

Свойства:

1. **Text** - надпись рядом с флажком (галочкой). Обычно вопрос.
2. **CheckState** - состояние контрола. (Отмечен/неотмечен/неопределено)
3. **ThreeState** - указывает позволять или нет выбирать третье состояние (Неопределенное)
4. **Checked** - возвращает true если контрол отмечен, false если не отмечен

Properties

checkBox1 System.Windows.Forms.CheckBox

FlatAppearance

FlatStyle	Standard
-----------	----------

Font

Font	Microsoft Sans Serif; 8,
ForeColor	ControlText
Image	(none)
ImageAlign	MiddleCenter
ImageIndex	(none)
ImageKey	(none)
ImageList	(none)
RightToLeft	No
Text	checkBox1
TextAlign	MiddleLeft
TextImageRelation	Overlay
UseMnemonic	True
UseVisualStyleBackCc	True
UseWaitCursor	False

Behavior

Data

Design

(Name)	checkBox1
GenerateMember	True
Locked	False
Modifiers	Private

Focus

CausesValidation	True
------------------	------

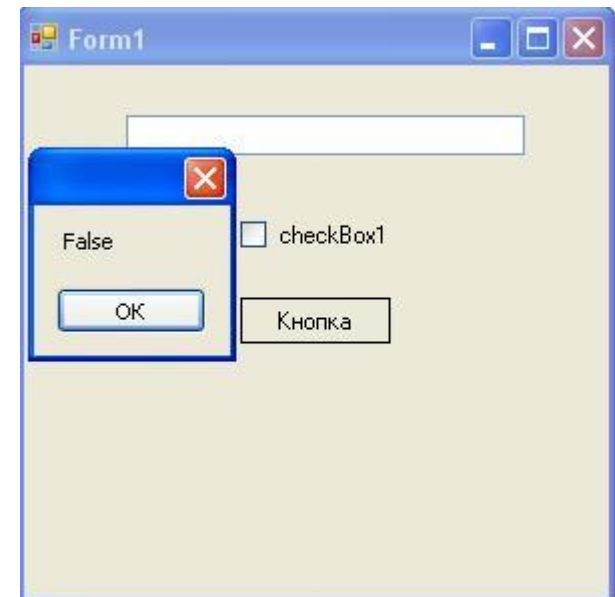
Text

The text associated with the control.

События CheckBox

- ▶ Основным событием является **CheckedChanged** - срабатывает когда изменено состояние контрола

```
private void checkBox1_CheckedChanged(object sender, EventArgs e)
{
    MessageBox.Show(checkBox1.Checked.ToString());
}
```



RadioButton

- ▶ Радио кнопки позволяют сделать выбор между несколькими предложенными вариантами. Чтобы компилятор знал, какие варианты ответа относятся к одному вопросу, радио кнопки объединяют в группу.

Свойства:

1. **Text** - Вариант ответа
2. **Checked** - Возвращает true если контрол отмечен, false если не отмечен

- ▶ Основным событием является

CheckedChanged – Отрабатывает
когда изменено состояние контрола



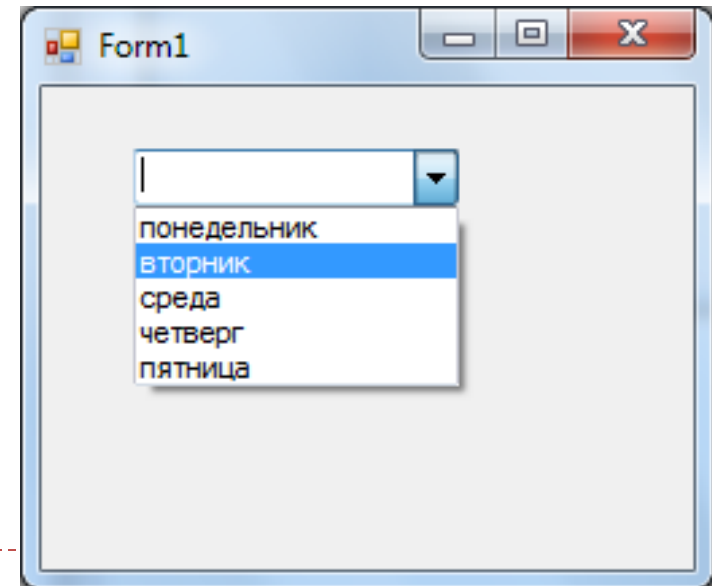
Выпадающий список ComboBox

► Свойства:

1. **Text** - Надпись на контроле. Надо понимать что это не элемент списка а всего лишь первоначальная надпись. И при выборе одного из элементов списка надпись больше не появится.
2. **Sorted** - Сортирует список
3. **Items** - Собственно сама коллекция
4. **DropDownStyle** - выбор стиля контрола

► Работа с элементами списка проходит через Items у которого есть методы

1. **Add** - Добавление элемента в список
2. **Clear** - Очищает список
3. **Remove** - Удаляет элемент списка



События ComboBox

- ▶ Для примера заполним список числами от 0 до 100

```
for (int i = 0; i < 100; i++)
```

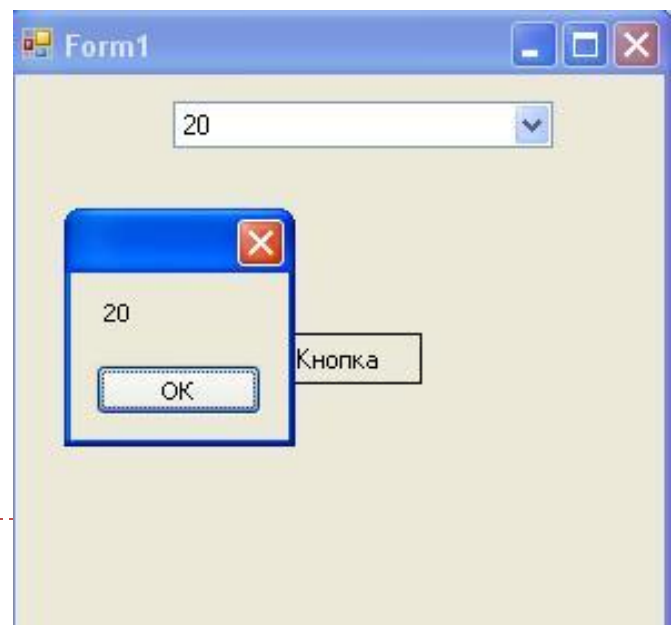
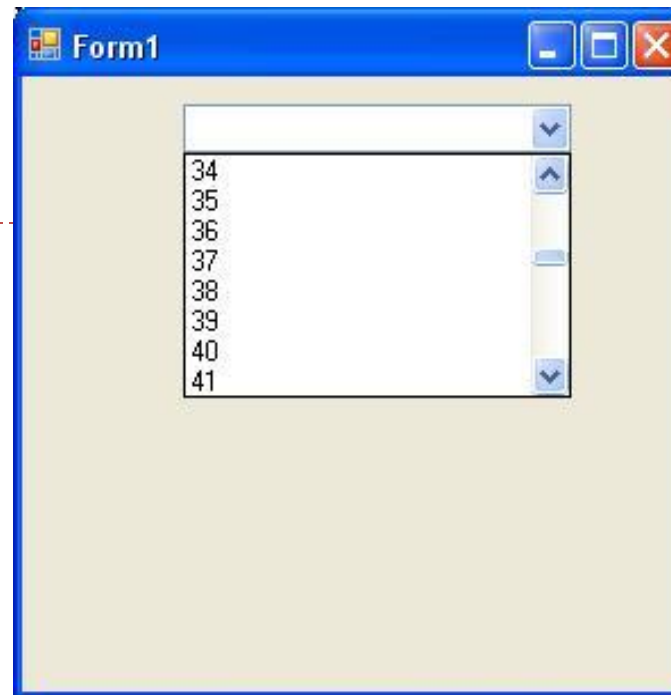
```
    comboBox1.Items.Add(i);
```

- ▶ Узнать выбранный элемент можно, используя свойства

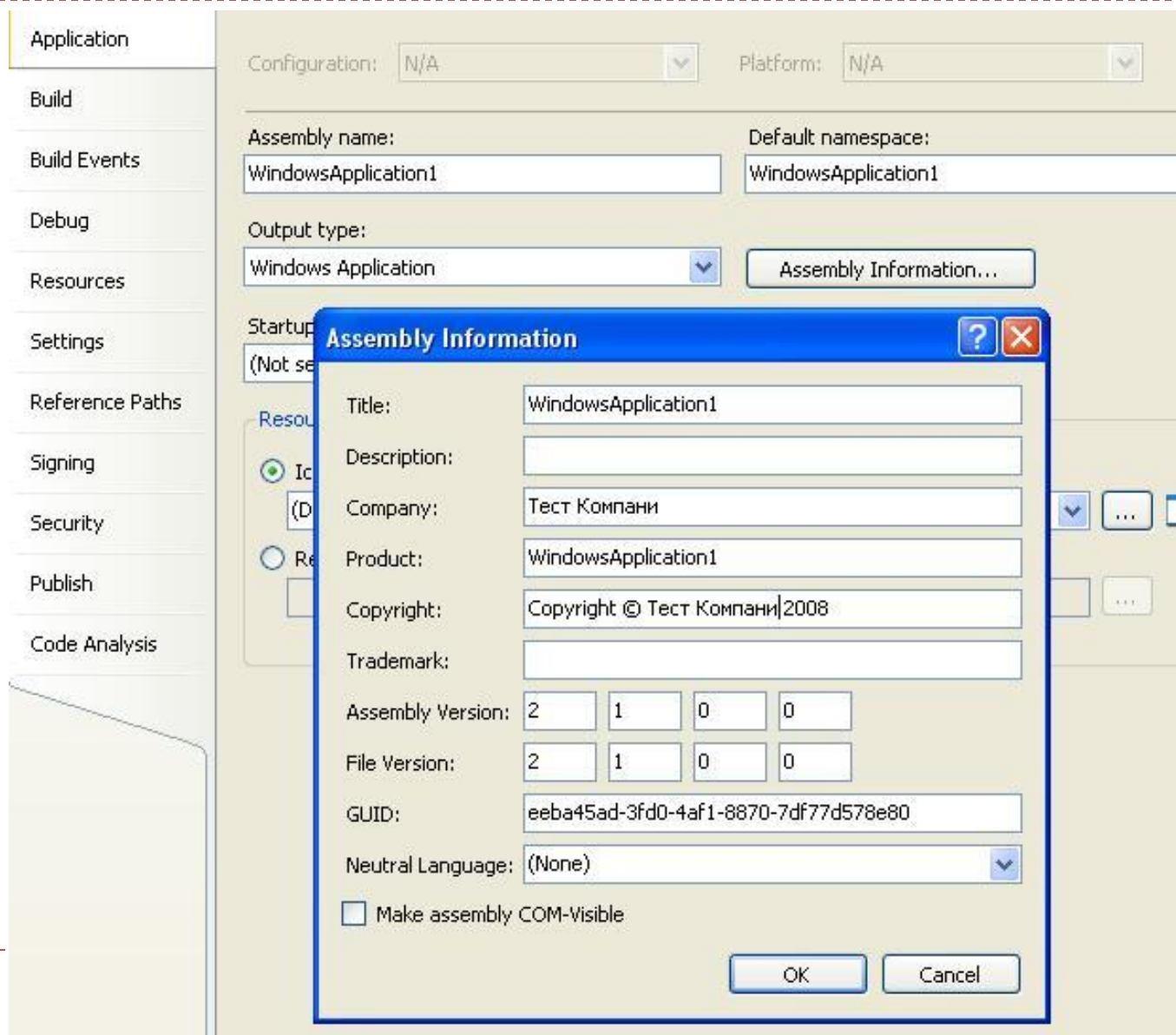
SelectedItem, SelectedText, SelectedIndex.

```
private void comboBox1_SelectedIndexChanged(object sender, EventArgs e){  
    MessageBox.Show(comboBox1.SelectedItem.ToString());  
}
```

Данный код помещен в обработчик события
SelectedIndexChanged который выполняется,
как только сменится элемент списка



Версия приложения



Версия приложения

- ▶ А теперь в обработчике Load формы выведем текущую версию в заголовок окна

```
private void Form1_Load(object sender, EventArgs e){  
    this.Text += "    Version:  " +  
System.Reflection.Assembly.GetExecutingAssembly().GetName().Version.ToString();  
}
```



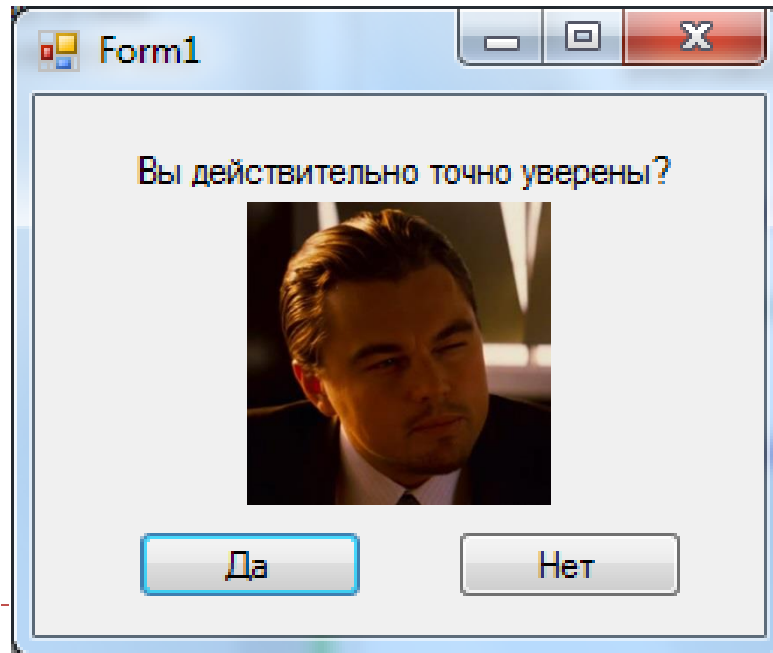
Таким образом из Assembly можно получить и другую информацию (компания, копирайт, и т.д.)

Модальные окна

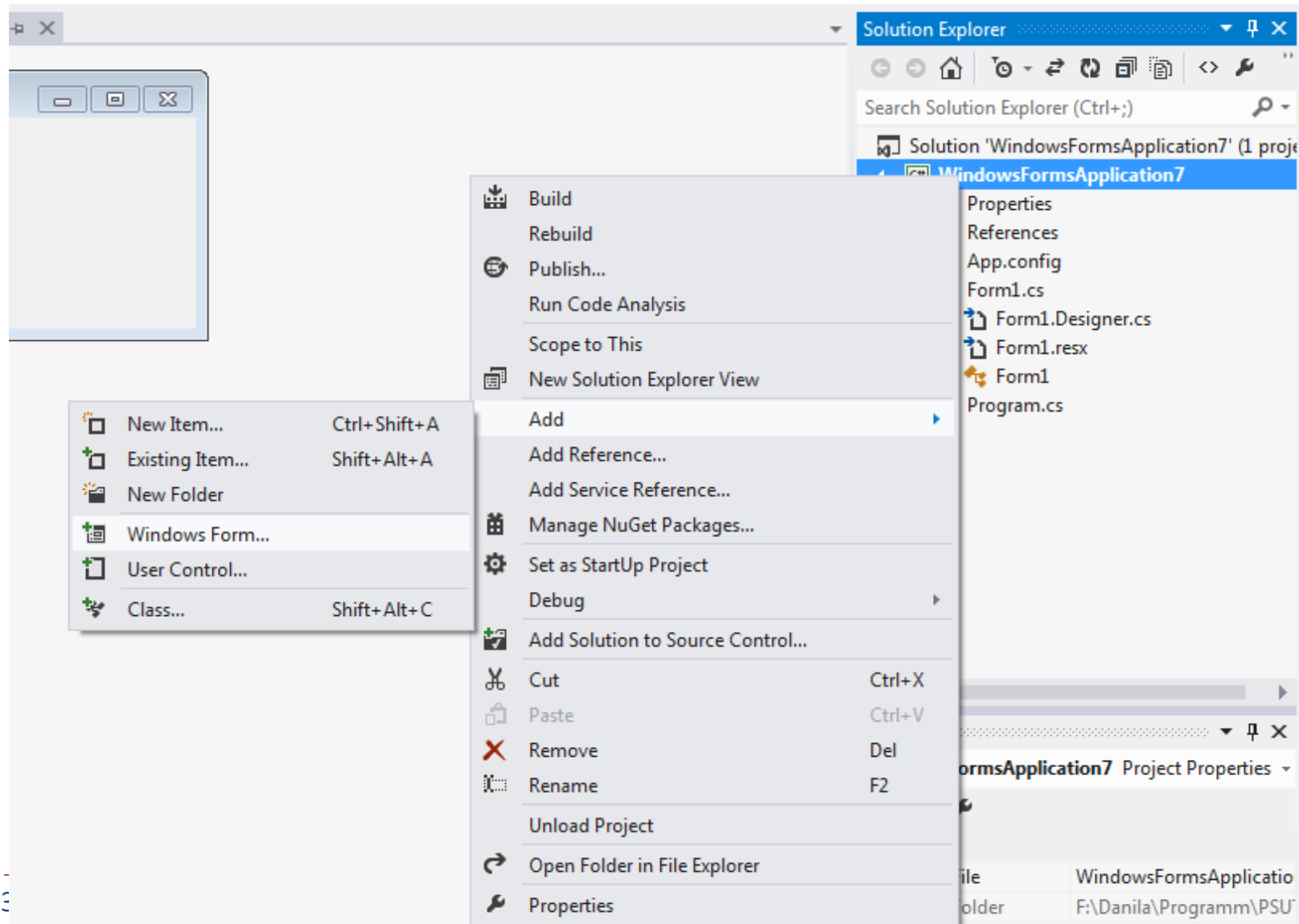
- ▶ Окно может быть как модальным так и обычным. Различие в том, что при **модальном** окне вы переключиться в другое окно пока данное (модальное) не закрыто.

Для управления показом существует у класса два метода:

- ▶ **Show()** - Отображает обычное окно (переключение между окнами возможно)
- ▶ **ShowDialog()** - Отображает диалоговое (модальное) окно (переключение между окнами не возможно)

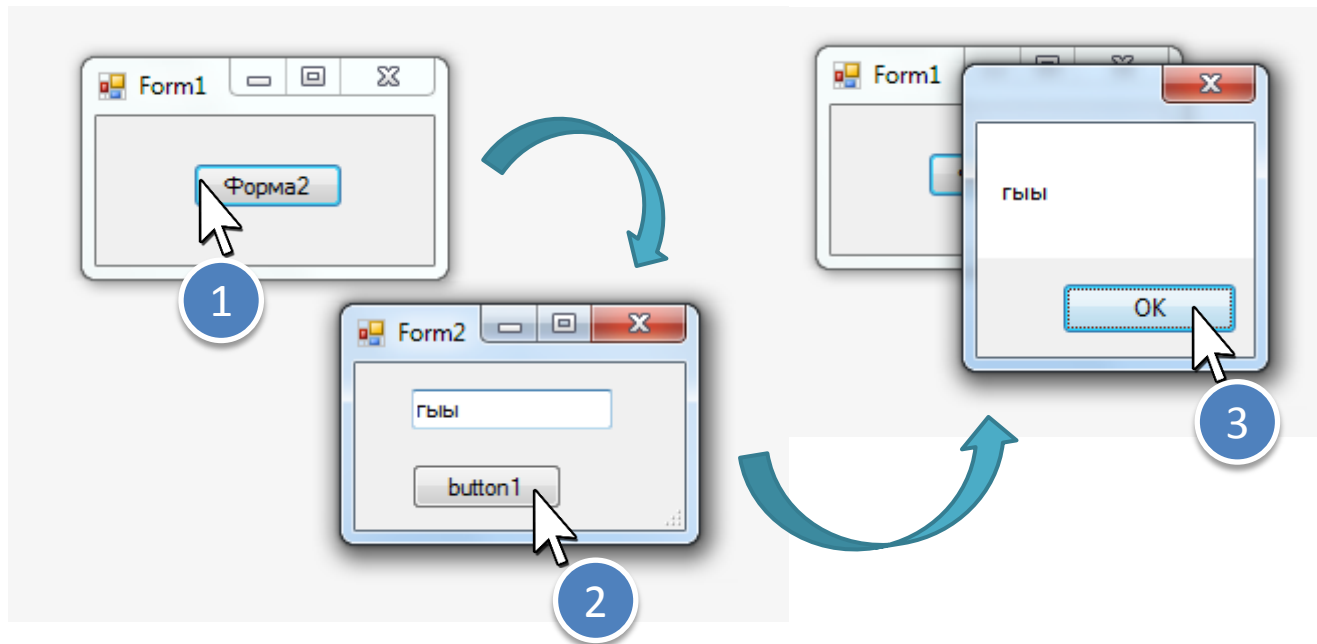


Добавление новой формы



Передача параметров между формами

- ▶ Рассмотрим пример передачи параметров из дочернего окна в родительское.
- ▶ В дополнение к нашей основной форме создадим небольшое окно Form2 с текстовым полем и кнопкой.



Передача параметров между формами

- ▶ На форме 2 (дочерней) установим свойство кнопки DialogResult OK, это будет означать, что при нажатии данной кнопки окно вернет DialogResult == OK, что и будет событием для отображения параметра, введенного в эту форму.
- ▶ Изменим режим доступа для текстового в файле Form2.Designer.cs:

```
public System.Windows.Forms.TextBox textBox1;
```

- ▶ На форме 1 (родительской) создадим кнопку вызова дочерней формы:

```
private void button1_Click(object sender, EventArgs e)
{
    Form2 window = new Form2();
    window.ShowDialog();
}
```

Добавим в этот метод отображение введенного в дочерней форме текста:

```
if (window.DialogResult == DialogResult.OK)
    MessageBox.Show(window.textBox1.Text);
```

Маска ввода MaskedTextBox.

- ▶ Контроль позволяет вводить текст по заранее заданному шаблону.
- ▶ Давайте добавим на форму этот компонент и создадим шаблон для ввода IP адреса (состоит из 4 групп (v. IP4) разделенных точками).
- ▶ Цифры отображаем нулями. Разделители точками. Нижнее поле показывает как будет отображаться контроль до ввода данных.

Select a predefined mask description from the list below or select Custom to define a custom mask.

Mask Description	Data Format	Validating Type
Numeric (5-digits)	12345	Int32
Phone number	(574) 555-0123	(none)
Phone number no area code	555-0123	(none)
Short date	12/11/2003	DateTime
Short date and time (US)	12/11/2003 11:20	DateTime
Social security number	000-00-1234	(none)
Time (European/Military)	23:20	DateTime
Time (US)	11:20	DateTime
Zip Code	98052-6399	(none)
<Custom>		(none)

Mask: 000.000.000.000 ☒ Use ValidatingType

Preview: _____._____._____._____

OK Cancel

Form1 Version: 2....

123.____.____.____

button1

Отображение ошибок ErrorProvider

- ▶ Компонент, который позволяет привлечь пользователя к какому либо критическому участку (выполнения действия или заполнения формы) – ErrorProvider.
- ▶ Проверим IP-адрес

```
private void button1_Click(object sender, EventArgs e) {  
    // код проверки IP  
    errorProvider1.SetError(maskedTextBox1, "Не верный IP");  
}
```



Достучаться до реестра

- ▶ Для работы с реестром в .NET используется пространство имен System.Microsoft.Win32 (using Microsoft.Win32;) класс RegistryKey
- ▶ Узнаем идентификатор продукта ОС Windows .

```
private void button1_Click_1(object sender, EventArgs e)
{
    RegistryKey localMachine = null;
    if (Environment.Is64BitOperatingSystem)
    {
        localMachine = RegistryKey.OpenBaseKey(Microsoft.Win32.RegistryHive.LocalMachine,
            RegistryView.Registry64);
    }
    else
    {
        localMachine = RegistryKey.OpenBaseKey(Microsoft.Win32.RegistryHive.LocalMachine,
            RegistryView.Registry32);
    }
    RegistryKey windowsNTKey = localMachine.OpenSubKey(@"Software\Microsoft\Windows" +
        " NT\CurrentVersion");
    MessageBox.Show(windowsNTKey.GetValue("ProductId").ToString());
}
```

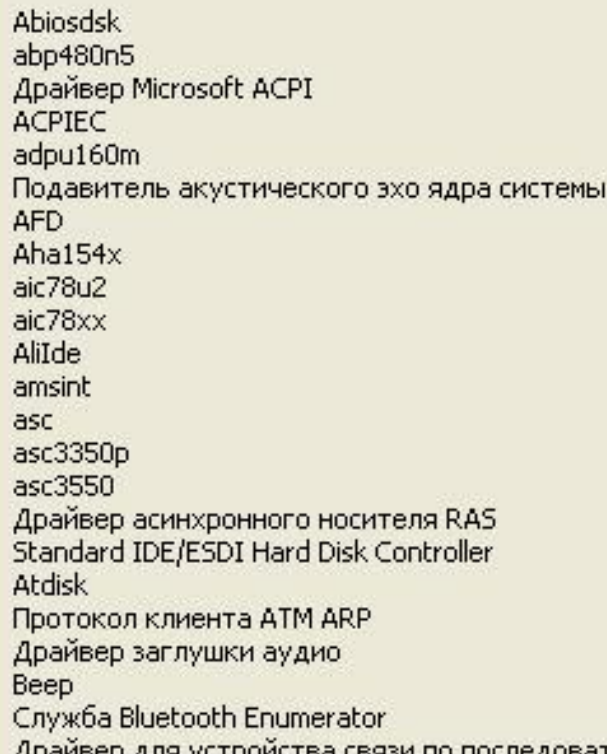


Достучаться до служб

- ▶ В .NET имеется пространство имен ServiceProcess (using System.ServiceProcess), которое позволяет через класс ServiceController работать со службами на данном компьютере. Пространство имен ServiceProcess сперва необходимо подключить через Обозреватель решений: правый клик на проекте – Add Reference (Добавить ссылку), выбрать в списке System.ServiceProcess.
- ▶ Получим список сервисов имеющихся на компьютере.

Для этого воспользуемся методом GetDevices() класса ServiceController.

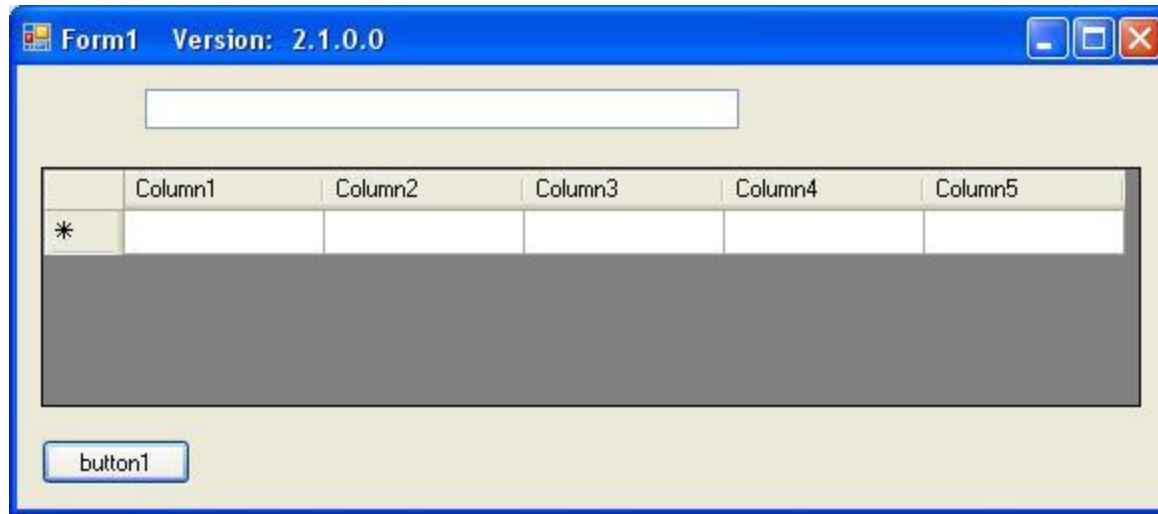
```
string services = "";  
foreach (ServiceController sc in  
ServiceController.GetDevices())  
    services += sc.DisplayName + "\n";  
MessageBox.Show(services);
```



```
Abiosdsk  
abp480n5  
Драйвер Microsoft ACPI  
ACPIEC  
adpu160m  
Подавитель акустического эхо ядра системы  
AFD  
Aha154x  
aic78u2  
aic78xx  
AliIde  
amsint  
asc  
asc3350p  
asc3550  
Драйвер асинхронного носителя RAS  
Standard IDE/ESDI Hard Disk Controller  
Atdisk  
Протокол клиента ATM ARP  
Драйвер заглушки аудио  
Beep  
Служба Bluetooth Enumerator  
Драйвер для устройства связи по последовательному каналу Bluetooth
```

Компоновка элементов

- Разместим компоненты TextBox и DataGridView как показано на рисунке



- Для того, чтобы грид растягивался во все стороны, необходимо привязать его ко всем сторонам формы, выставив свойство **Anchor** в Top, Bottom, Left, Right
- Что бы грид не закрывал кнопку при увеличении, необходимо изменить Anchor для кнопки на значение Bottom, Left.

Считываем нажатую клавишу

- ▶ Создадим в событиях формы (окно свойств), метод, обрабатывающий нажатия клавиш (событие KeyPress):

```
private void Form1_KeyPress(object sender, KeyPressEventArgs e) {}
```

- ▶ В сгенерированной функции передаются аргументы связанные с нажатой кнопкой.
- ▶ Добавим вывод нажатой кнопки в метку на форме

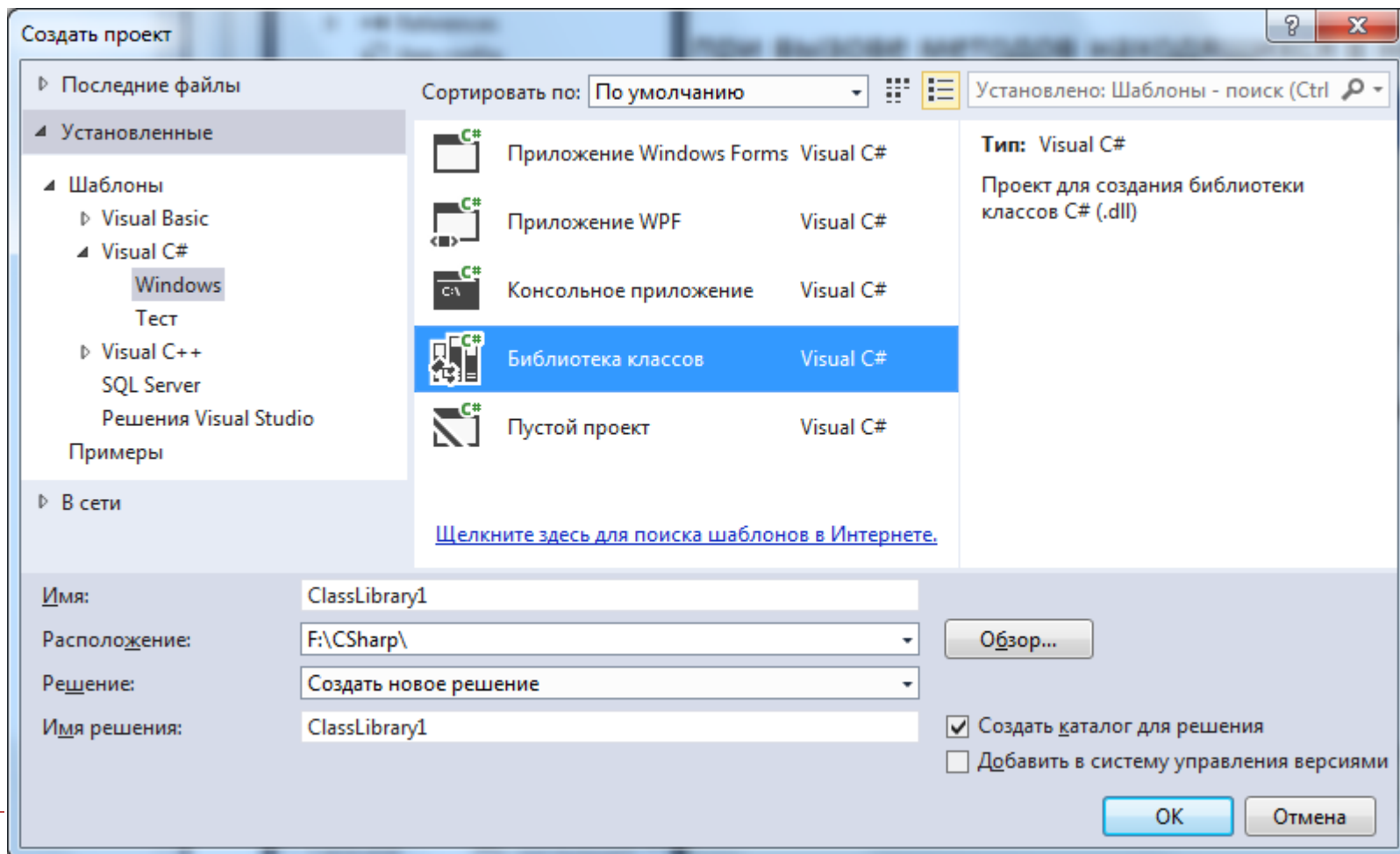
```
label1.Text = e.KeyChar.ToString();
```



Динамическая библиотека DLL

- ▶ DLL загружается в память только при вызове методов находящихся в ней.
- ▶ Создаем

проект



Динамическая библиотека DLL

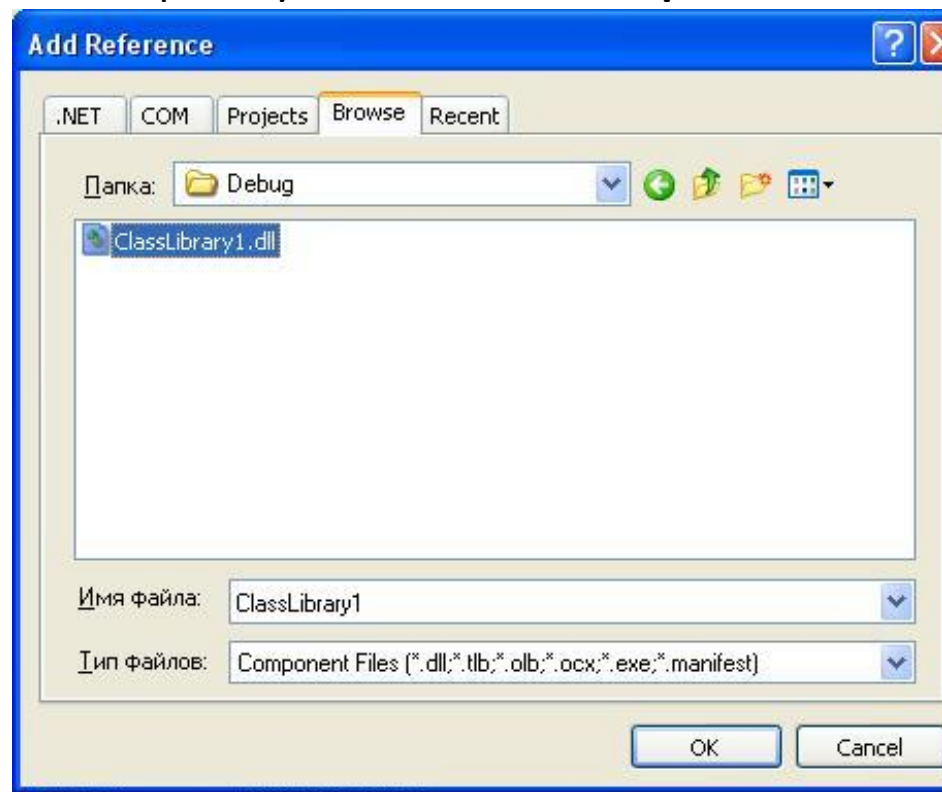
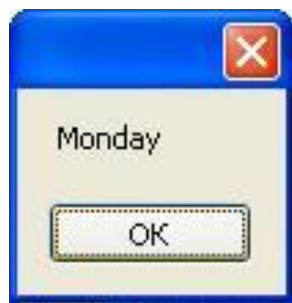
- ▶ Создадим метод который будет возвращать текущий день недели

```
using System;
using System.Collections.Generic;
using System.Text;
namespace ClassLibrary1{
    public class Class1{
        public static string Today(){
            return DateTime.Now.DayOfWeek.ToString();
        }
    }
}
```

- ▶ Скомпилируем DLL

Динамическая библиотека DLL

- ▶ В проекте Кликаем правой кнопкой мыши по проекту в окне **SolutionExplorer**
- ▶ Нажимаем **Add Reference**
- ▶ Ищем .dll и добавляем ее в проект
- ▶ После этих действий пространство имен ClassLibrary1 и класс Class1 будут видны в нашем проекте

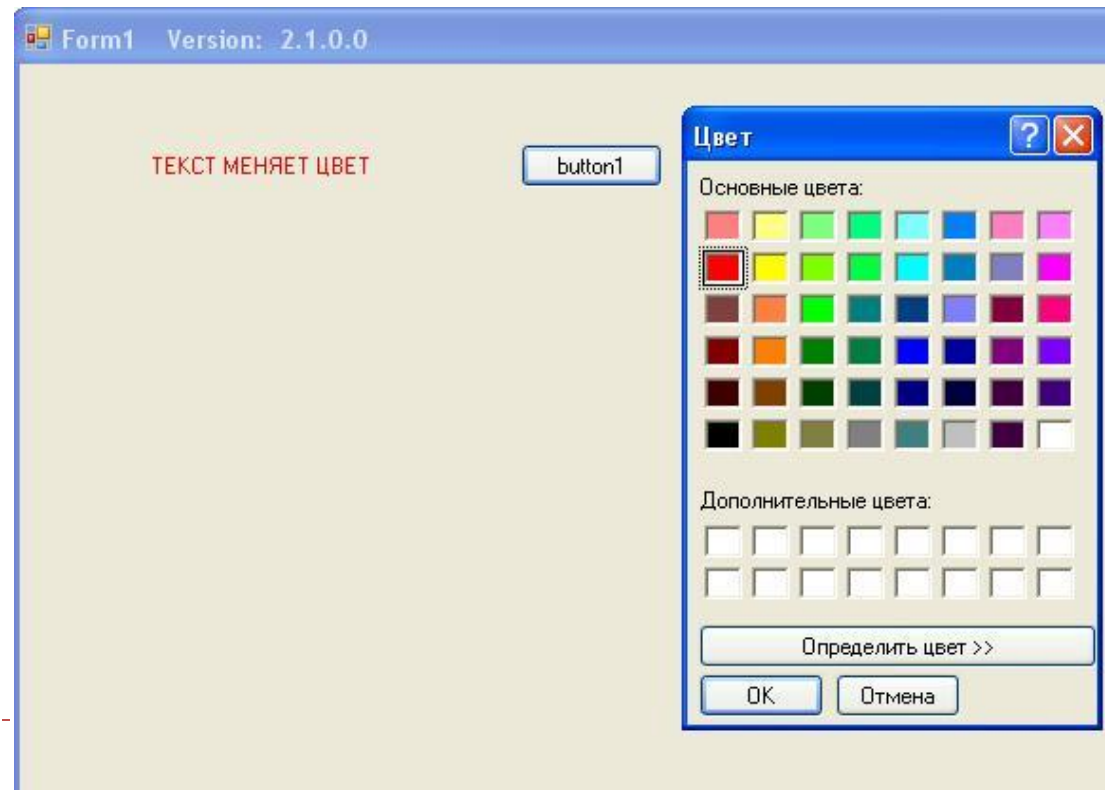


- ▶ Вызовем единственный метод класса
`MessageBox.Show(ClassLibrary1.Class1.Today());`

Выбираем цвет - ColorDialog

- ▶ На примере мы будем разукрашивать надпись в Label цветом выбранным в диалоговом окне. Создаем colorDialog и бросаем его на форму.
- ▶ При нажатии на кнопке открываем диалог цветов. При положительном выборе цвета в диалоговом окне, назначаем этот цвет метке (Label1).

```
private void button1_Click_1(object sender, EventArgs e){  
    DialogResult dr =  
    colorDialog1.ShowDialog();  
    if (dr == DialogResult.OK)  
        label1.ForeColor =  
        colorDialog1.Color;  
}
```



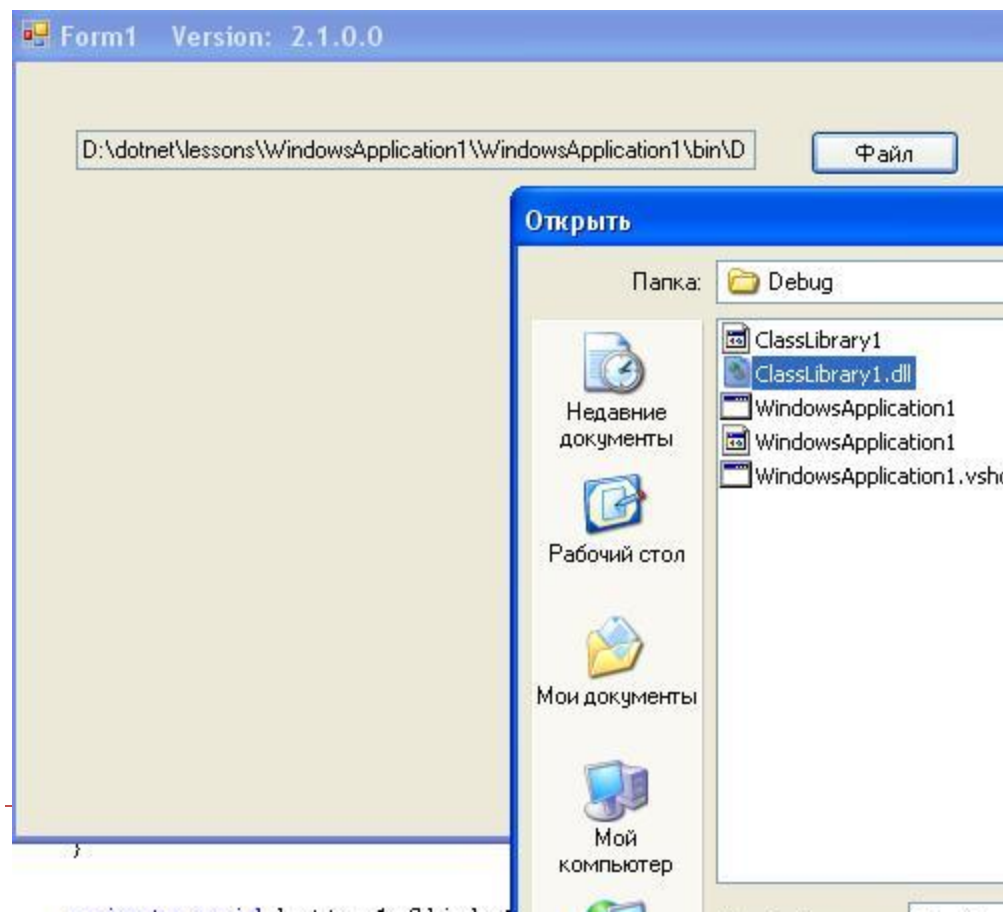
Файловый диалог - OpenFileDialog

- ▶ Позволяет выбрать файл из имеющихся на компьютере.
- ▶ Добавляем компонент на форму, для вывода полного пути до файла разместим textbox, на кнопку устанавливаем открытие формы и обработку DialogResult

```
private void button1_Click_1
(object sender, EventArgs e)X
    DialogResult dr =
        openFileDialog1.ShowDialog();
    if (dr == DialogResult.OK)
        textBox1.Text =
            openFileDialog1.FileName;
}
```

Свойства OpenFileDialog

- ▶ **Title** - задает заголовок окна
- ▶ **Filter** - Позволяет фильтровать файлы по расширению (C# файлы | *.cs)
- ▶ **MultiSeelct** - Позволяет выбирать несколько файлов
- ▶ **initialDirectory** - Начальный каталог



Компонент WebBrowser

- ▶ Добавляем текстовое поле, в которое будем вводить URL.
- ▶ Добавляем кнопку и используем метод Navigate для отображения в браузере

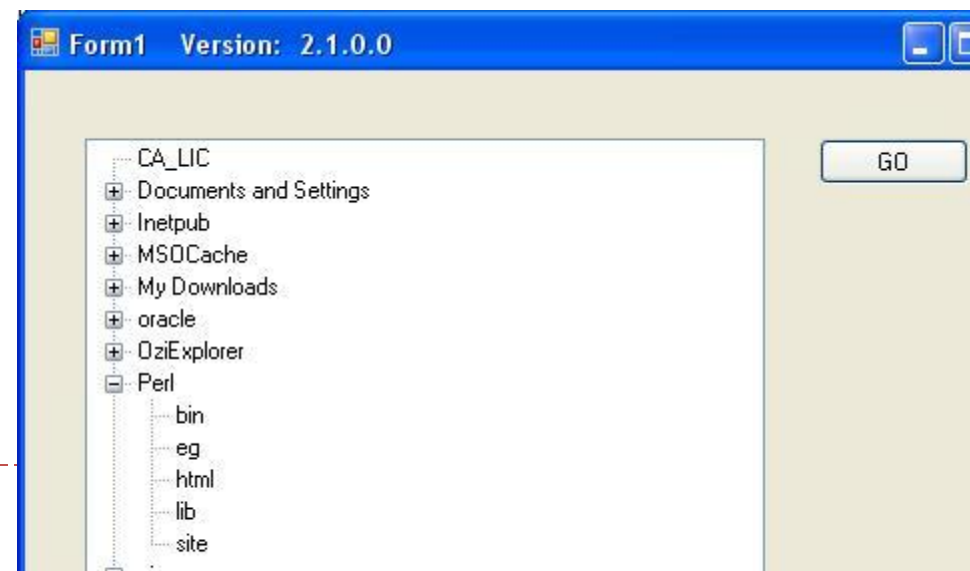
```
private void button1_Click_1(object sender, EventArgs e)
{
    webBrowser1.Navigate("http://" + textBox1.Text);
}
```



Динамическое создание дерева – TreeView

- ▶ Получим структуру двух первых уровней каталогов вашего диска C. Для этого нам потребуется класс DirectoryInfo из пространства имен System.IO
- ▶ Используя метод GetDirectories мы получим список каталогов, далее будем рекурсивно добавлять узлы в дерево используя метод Add

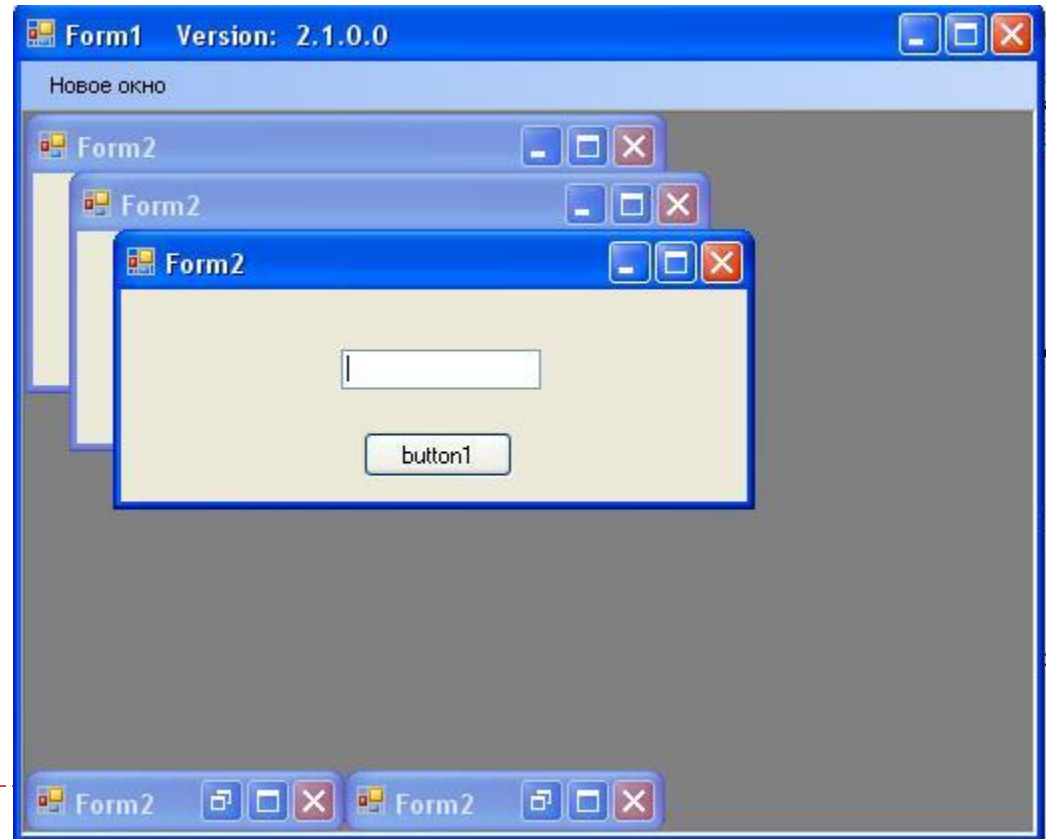
```
DirectoryInfo di = new DirectoryInfo(@"c:\");  
foreach (DirectoryInfo di2 in di.GetDirectories())  
{  
    TreeNode tn = new TreeNode(di2.Name);  
    DirectoryInfo d3 = new DirectoryInfo(di2.FullName);  
    foreach (DirectoryInfo d4 in d3.GetDirectories())  
    {  
        tn.Nodes.Add(new TreeNode(d4.Name));  
    }  
    treeView1.Nodes.Add(tn);  
}
```



Окна родитель и потомок

- ▶ Для получения вложенной структуры окон (например как в Word'e) необходимо в родительском окне указать свойство `IsMdiContainer = true`
- ▶ Чтобы все окна отображались как вложенные, необходимо у создаваемых окон указать в свойстве `MdiParent` родительское окно

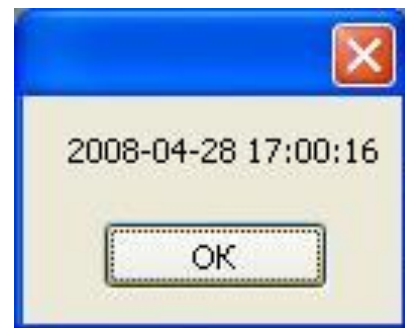
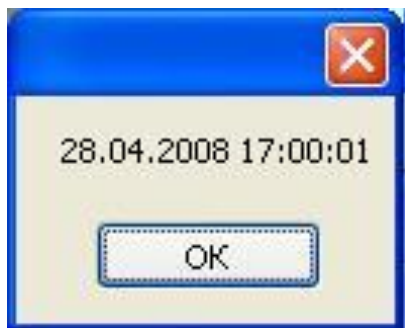
```
Form2 w = new Form2();  
w.MdiParent = this;  
w.Show();
```



Управляем форматом даты

- ▶ По умолчанию все настройки отображения даты, время, разделителей дробных чисел берутся из региональных настроек Windows.
- ▶ Для переопределения отображения даты создадим класс DateTimeFormatInfo из пространства имен System.Globalization и настроим его свойство ShortDatePattern в тот формат который нам нужен.

```
MessageBox.Show(DateTime.Now.ToString());  
  
    System.Globalization.DateTimeFormatInfo dtfi = new  
System.Globalization.DateTimeFormatInfo();  
    dtfi.ShortDatePattern = "yyyy-MM-dd";  
    MessageBox.Show(DateTime.Now.ToString(dtfi));
```



Региональные параметры

- ▶ При попытке выполнить преобразование строки в `double` следующей командой
`Convert.ToDouble("2.5");`

на русской ОС в процессе выполнения появится ошибка о несоответствии формата.

- ▶ При преобразовании берутся региональные настройки, а для России в качестве разделителя целой и дробной части по умолчанию запятая, а в тексте точка.
- ▶ Для корректного преобразования необходимо передать провайдер формата.
- ▶ Класс **NumberFormatInfo** служит как раз для этого

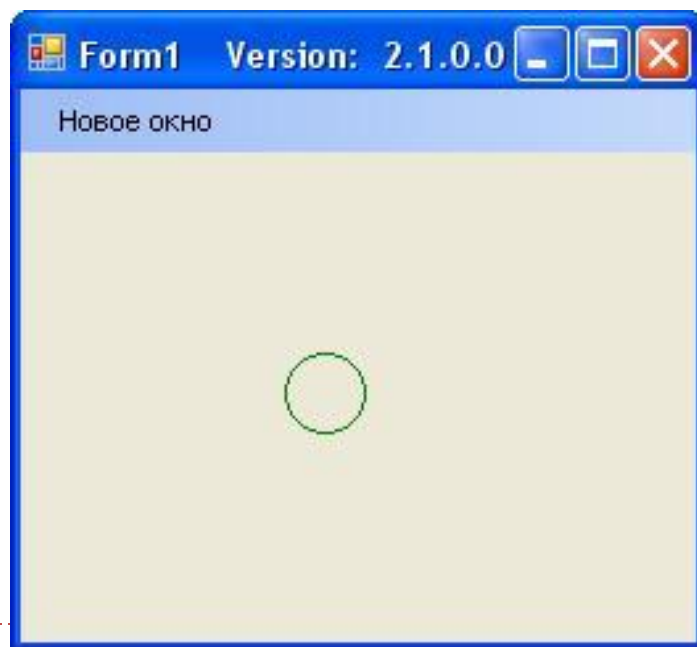
```
System.Globalization.NumberFormatInfo nfi = new  
System.Globalization.NumberFormatInfo();  
nfi.NumberDecimalSeparator = ".";  
Convert.ToDouble("2.5", nfi);
```

Рисунки на форме. GDI+

- ▶ Для примера нарисуем на нашей форме круг, используя класс Graphics и метод DrawEllipse:

```
Graphics g = Graphics.FromHwnd(this.Handle);  
g.DrawEllipse(new Pen(Color.Green), 100, 100, 30, 30);
```

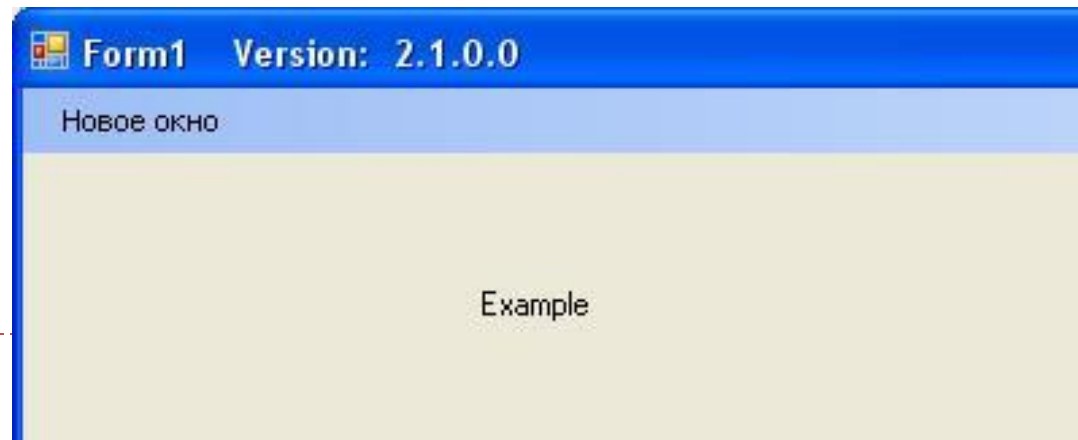
- ▶ В качестве параметров методу передаются: объект карандаш, координаты расположения и размер по вертикали и горизонтали.



Языковая кастомизация

- ▶ В .NET контролы могут хранить текст в нескольких языках.
- ▶ Рассмотрим возможности на примере метки Label.
- ▶ У формы, на которой будет использоваться мультиланг, устанавливаем у свойства Language язык - русский .
- ▶ Пишем свойство метки Text - "Пример".
- ▶ Меняем Language у формы на английский , далее у метки Text на "Example".
- ▶ В конструкторе формы поменяем язык для UserInterface на английский и посмотрим что будет написано в метке

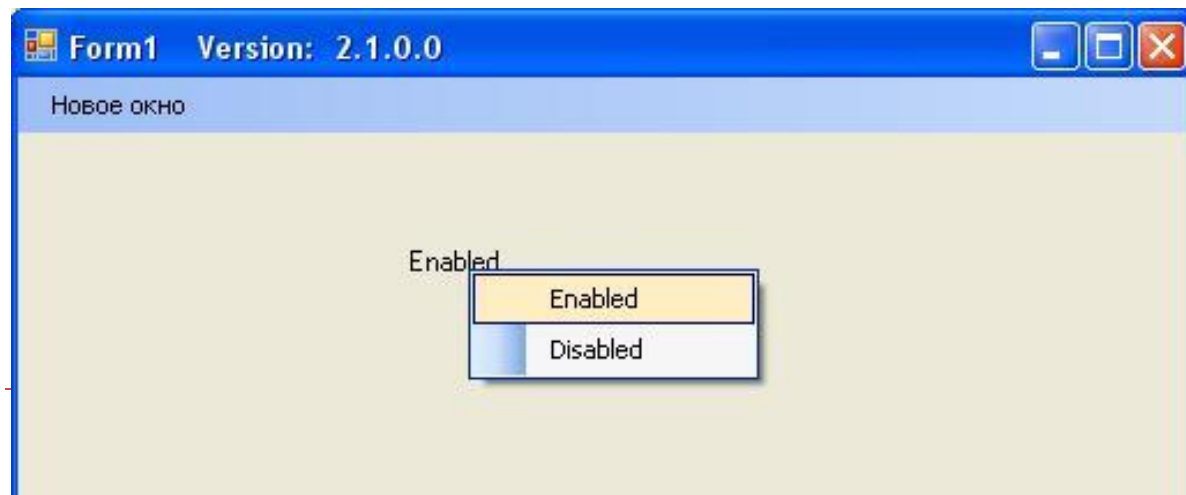
```
public Form1()  
{  
    System.Threading.Thread.CurrentThread.CurrentUICulture = new  
System.Globalization.CultureInfo("en-US");  
    InitializeComponent();  
}
```



Контекстное меню

- ▶ Добавим контекстное меню к одному из контролов, а именно к метке Label.
- ▶ Необходимо на форме разместить компонент ContextMenuStrip.
- ▶ Добавляем в него два пункта "Enabled" и "Disabled". В свойстве contextMenuStrip у метки указываем имя нашего меню. При щелчке правой кнопкой по Label покажется наше меню с двумя пунктами. Построим обработчик для каждого пункта, по щелчке по которому будет меняться текст у метки

```
private void enabledToolStripMenuItem_Click(object sender, EventArgs e) {  
    label1.Text = "Enabled";  
}  
  
private void disabledToolStripMenuItem_Click(object sender, EventArgs e) {  
    label1.Text = "Disabled";  
}
```



Значок в трее - NotifyIcon

- ▶ Для отображения в трее необходимо указать иконку для отображения в свойстве Icon.
- ▶ Одной из привлекательных возможностей данного компонента является возможность показывать всплывающие уведомления в области трее.
- ▶ Для этого используется метод ShowBalloonTip. В качестве параметров принимает время в миллисекундах на которое покажется подсказка, заголовок, текст и иконка уведомления

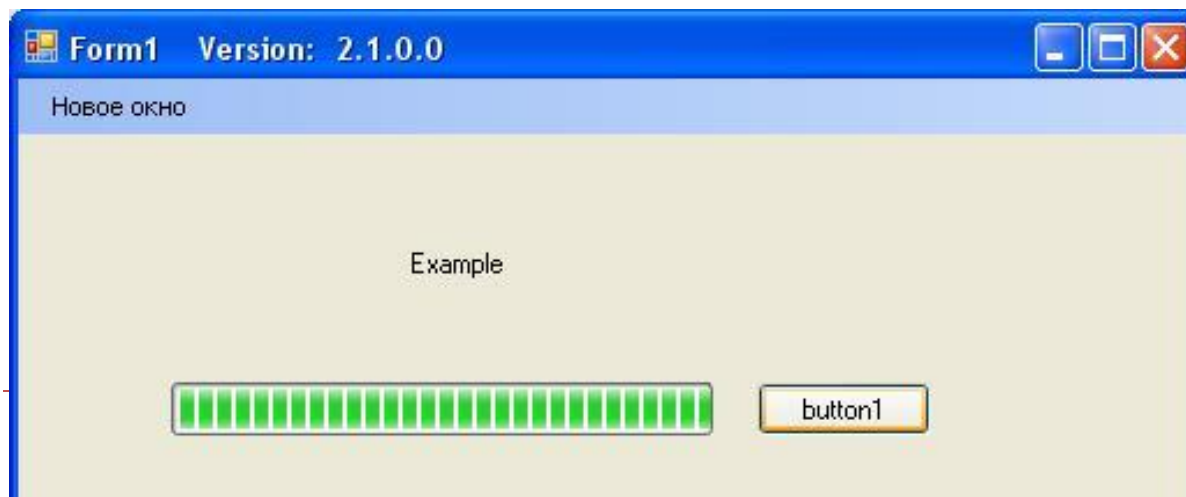
```
notifyIcon1.ShowBalloonTip(5000, "Title", "Text", ToolTipIcon.Info);
```



Ход процесса - ProgressBar

- ▶ Основными свойствами является нижний и верхний предел допустимых значений - Minimum и Maximum. Value содержит текущее значение прогресс бара.
- ▶ Также можно задать стиль отображения контрола в свойстве Style.
- ▶ На примере покажем как прокрутить прогресс бар от 0 до 100

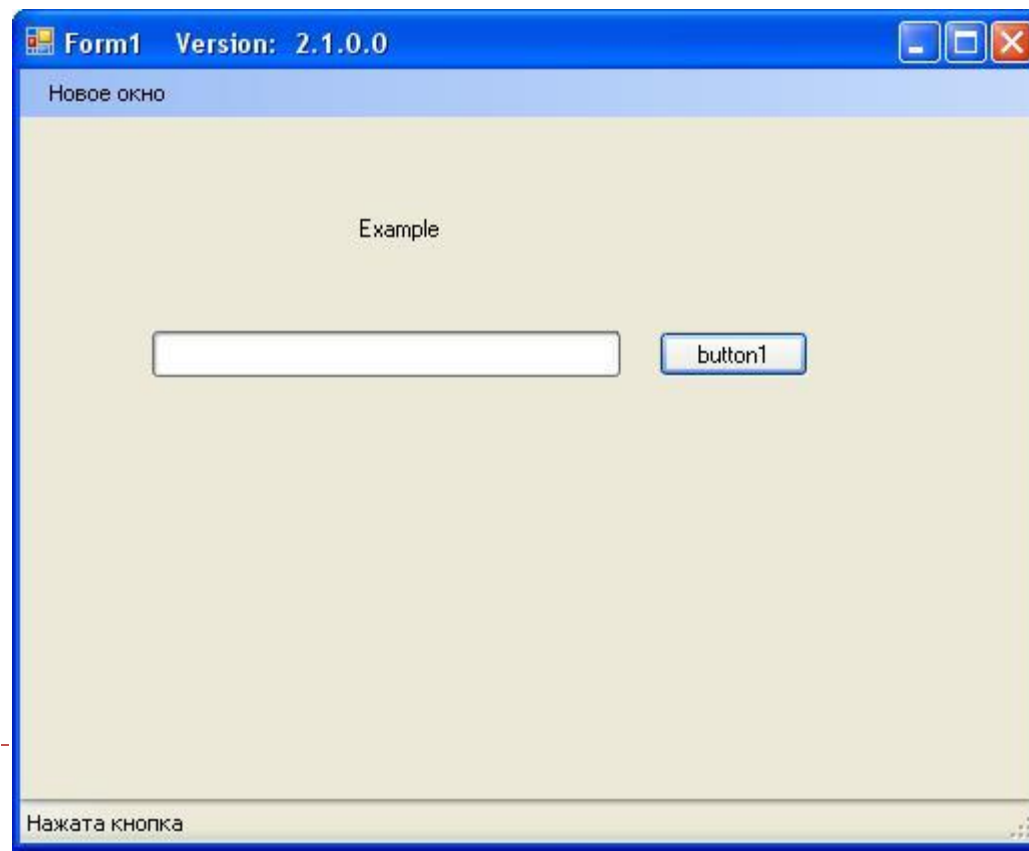
```
for (int i = progressBar1.Minimum; i < progressBar1.Maximum; i++)  
{  
    progressBar1.Value = i;  
    System.Threading.Thread.Sleep(10);  
}
```



Строка состояния - StatusStrip

- ▶ На самом деле этот компонент - контейнер и может кроме текстовой строки отображать также прогрессбар, splitbutton, dropdownbutton.
- ▶ Выберите необходимый элемент для отображения (допустим, toolStripStatusLabel)
- ▶ Для вывода информации в него используется свойство Text.

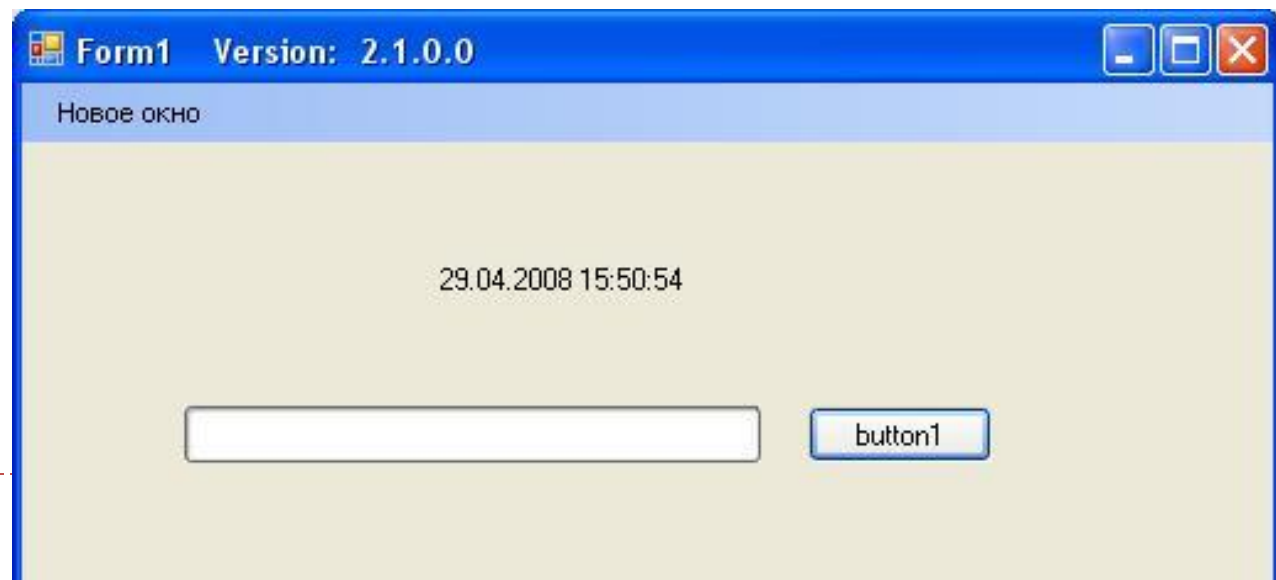
```
toolStripStatusLabel1.Text = "Нажата кнопка";
```



Таймер

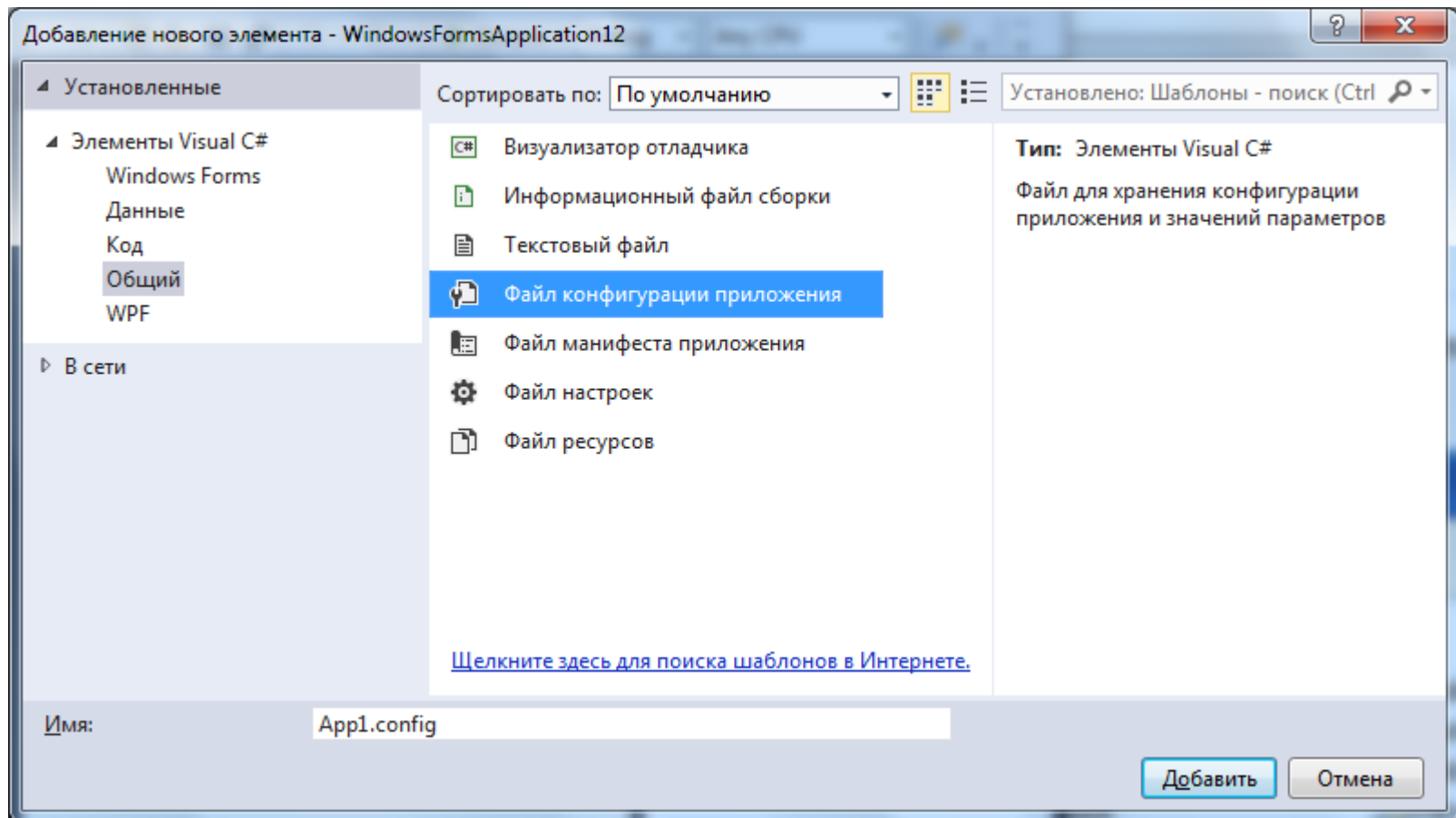
- ▶ Timer используется для запуска каких либо действий через определенное время.
- ▶ Для задания времени срабатывания используется свойство interval.
- ▶ Существует единственно событие Tick которое отрабатывает через время заданное в Interval.
- ▶ Можно временно выключать таймер переводя свойство enable в false.

```
private void timer1_Tick(object sender, EventArgs e)
{
    label1.Text = DateTime.Now.ToString();
}
```



Работа с конфигом

- ▶ Рассмотрим механизм работы с конфигурационными файлами в WIN приложениях.
- ▶ Для начала добавляем в проект конфиг файл



Работа с конфигом

- ▶ Будет создан файл в XML формате. Подключаем библиотеку

```
using System.Configuration;
```

- ▶ Создаем объект через который будем взаимодействовать с файлом конфигурации

```
System.Configuration.Configuration config =  
ConfigurationManager.OpenExeConfiguration(ConfigurationUserLevel.None);
```

- ▶ Выбираем секцию конфиг файла

```
AppSettingsSection appSettings =  
(AppSettingsSection)config.GetSection("appSettings");
```

- ▶ Когда секция подключена можно читать и сохранять параметры

- ▶ Читаем параметры конфиг файла

```
MessageBox.Show(appSettings.Settings["server"].Value);
```

- ▶ Записываем новое значение

```
appSettings.Settings["server"].Value = "NewServer";
```

- ▶ После внесения новых значений необходимо сохранить конфиг

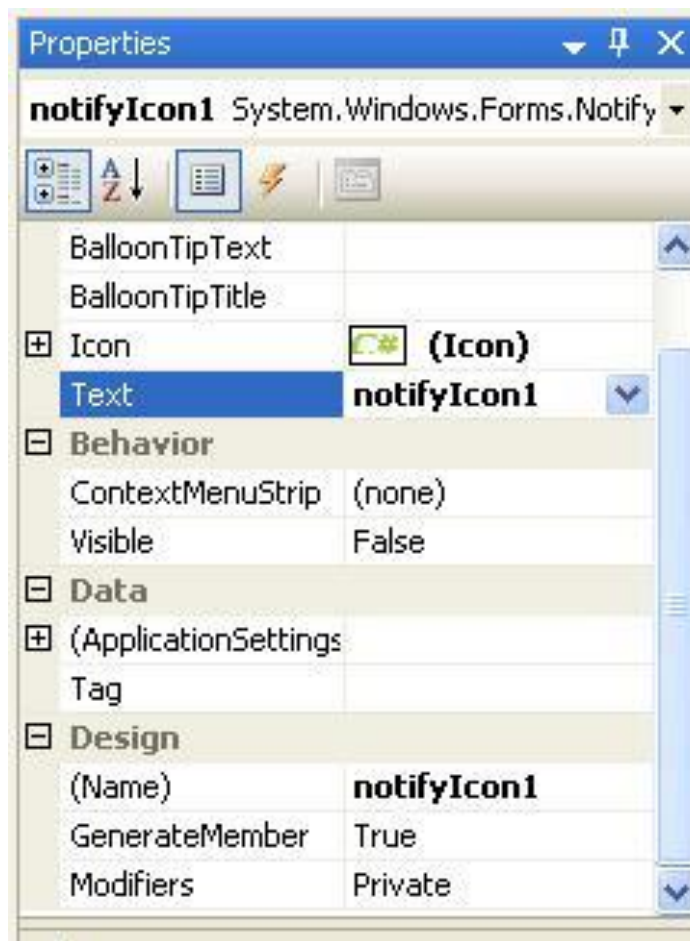
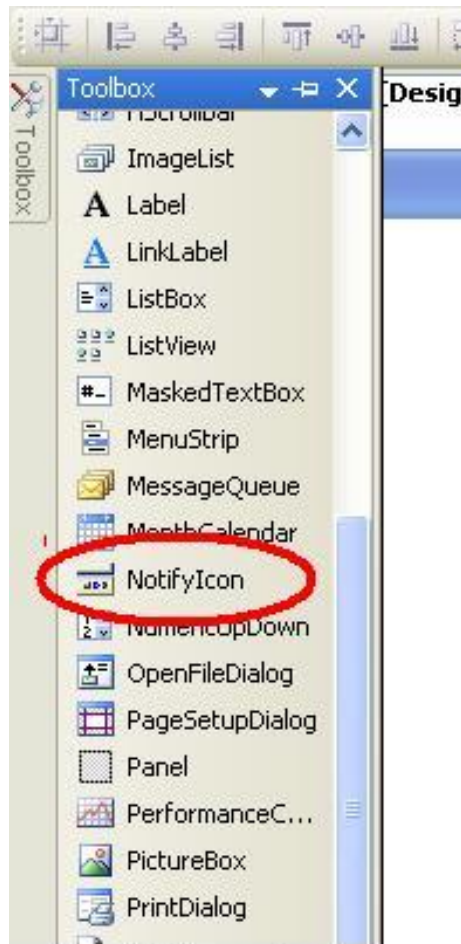
```
config.Save();
```

Работа с конфигом

- ▶ Пример файла конфигурации

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <sectionGroup name="applicationSettings"
type="System.Configuration.ApplicationSettingsGroup, System,
Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" >
      <section name="ProjectReport.Properties.Settings"
type="System.Configuration.ClientSettingsSection, System, Version=2.0.0.0,
Culture=neutral, PublicKeyToken=b77a5c561934e089" requirePermission="false"
/>
    </sectionGroup>
  </configSections>
  <appSettings>
    <add key="server" value="MyServer" />
  </appSettings>
  <connectionStrings />
</configuration>
```

Свернуть окно в трей - NotifyIcon



Свернуть окно в трей - NotifyIcon

- ▶ За сворачивание окна отвечает событие Deactivate
- ▶ Проверяем состояние окна, если оно действительно свернуто, то скрываем его из области отображения в эксплорере и показываем нашу иконку в трее

```
private void Form1_Deactivate(object sender, EventArgs e){  
    if (this.WindowState == FormWindowState.Minimized){  
        this.ShowInTaskbar = false;  
        notifyIcon1.Visible = true;  
    }  
}
```

- ▶ Восстановить окно можно по событию клика по иконке, либо можно привязать контекстное меню.

```
private void notifyIcon1_Click(object sender, EventArgs e){  
    if (this.WindowState == FormWindowState.Minimized){  
        this.WindowState = FormWindowState.Normal;  
        this.ShowInTaskbar = true;  
        notifyIcon1.Visible = false;  
    }  
}
```

Рекомендации

Видеоурок - основы растягиваемого интерфейса от
Глеба Горелова:

- ▶ <http://youtu.be/yyNyX823v0I>

