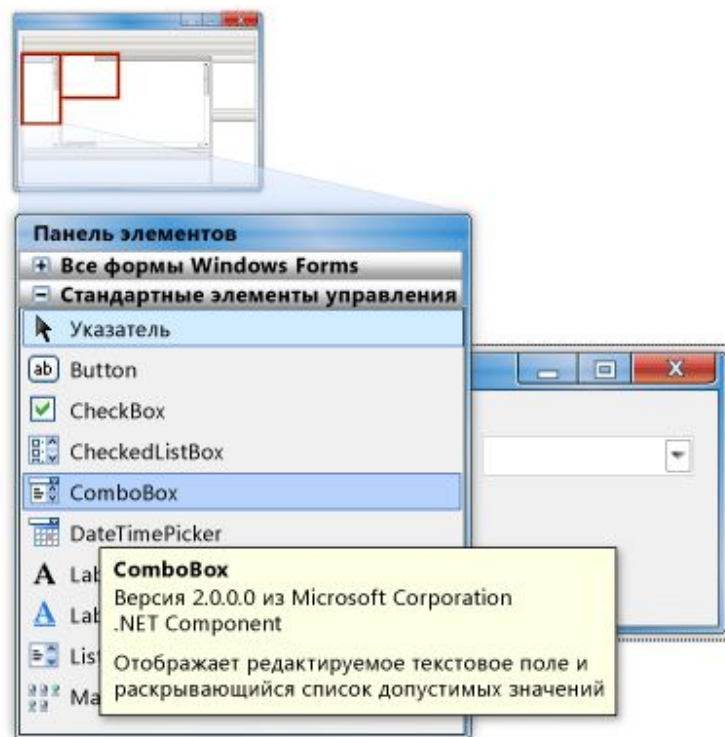


# Свойства объектов

# Панель элементов

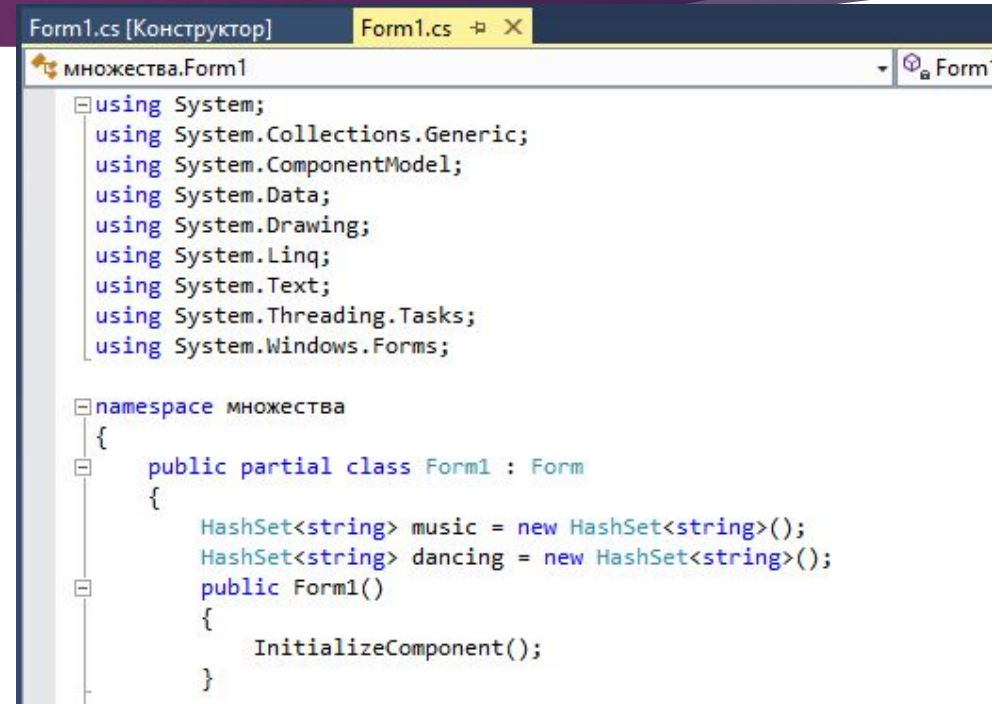
- ▶ Панель элементов представляет собой контейнер для всех элементов управления, которые можно добавить в приложение Windows Forms. По умолчанию Панель элементов находится с левой стороны в интерфейсе IDE.



Можно настроить панель элементов так, чтобы отображать их все (тогда список будет длиннее и тяжелее найти нужный элемент), а можно – только стандартные.

# Окно редактора кода

- ▶ Текстовый редактор - основной инструмент программиста. Вызывается командой «Перейти к коду» контекстного меню формы или командой «Код» меню «Вид». Для каждого элемента проекта (формы, программного модуля) открывается отдельная вкладка в окне редактора кода.



```
Form1.cs [Конструктор]  Form1.cs ▢ X
множества.Form1
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace множества
{
    public partial class Form1 : Form
    {
        HashSet<string> music = new HashSet<string>();
        HashSet<string> dancing = new HashSet<string>();
        public Form1()
        {
            InitializeComponent();
        }
    }
}
```

# Свойства объектов форм

- ▶ У каждого объекта есть свои свойства и методы, но есть и общие, которые есть практически у всех объектов.
- ▶ Свойства объектов можно изменять в коде, а можно на панели свойств

# Общие свойства:

- ▶ **Font** - Стиль шрифта. Если значение свойства Font изменить у формы, то значение этого свойства изменится у всех объектов находящихся на форме.
- ▶ **ForeColor** - Цвет шрифта. Если значение свойства ForeColor изменить у формы, то значение этого свойства изменится у всех объектов находящихся на форме.
- ▶ **Cursor** - в значение этого свойства надо выбрать курсор, который будет появляться при передвижении мыши над объектом
- ▶ **Location** - Координаты объекта
- ▶ **Size** - Размеры объекта, Height - высота, Width – длина
- ▶ **Text** - Текст, который будет на объекте. По умолчанию значение этого свойства - имя объекта.

# Свойства формы

- ▶ **AcceptButton** - Если на форме расположена кнопка, то ее можно будет указать в этом свойстве, и тогда, при загрузке формы, она будет выделена
- ▶ **ControlBox** - Если значение этого свойства равно `True`, то в заголовке формы будут видны 3 кнопки (заккрыть, развернуть, свернуть), если `False`, то эти кнопки видны не будут.
- ▶ **FormBorderStyle** - при помощи этого свойства можно изменить бордюр формы, убрать заголовков, сделать чтобы размеры формы нельзя было изменить
- ▶ **Icon** - при помощи этого свойства можно установить иконку для формы(и для программы), но об этом потом
- ▶ **MaximizeBox** - Если значение равно `False`, то кнопка 'Развернуть' будет заблокирована, если `True`, то разблокирована.
- ▶ **MaximumSize** - При помощи этого свойства можно установить максимальные размеры формы, на которые ее можно растянуть
- ▶ **MinimizeBox** - Если значение равно `False`, то кнопка 'Свернуть' будет заблокирована, если `True`, то разблокирована.

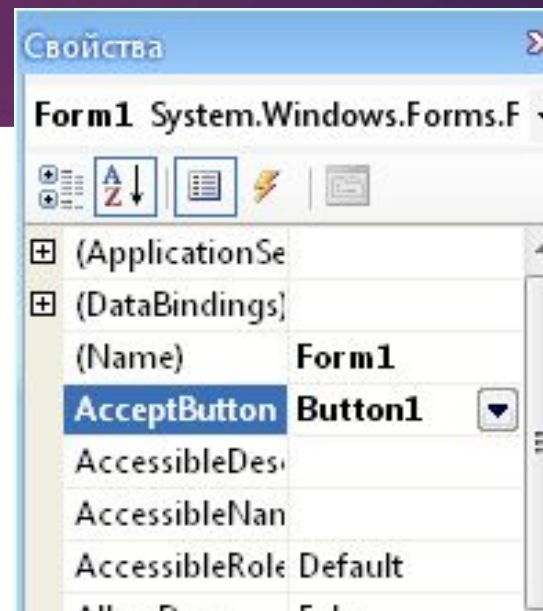
# Объект Form имеет свойства AcceptButton и CancelButton

## AcceptButton

- Возвращает или задает кнопку на форме, которая срабатывает при нажатии клавиши ENTER.

## CancelButton

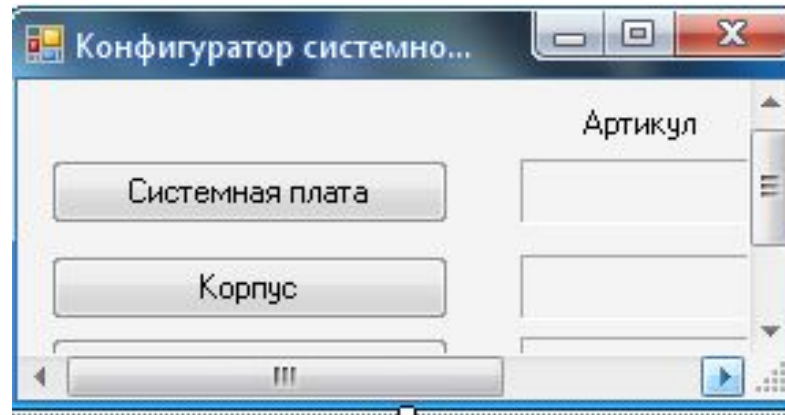
- Возвращает или задает кнопку, которая срабатывает при нажатии клавиши ESC.



Указанные в свойствах формы клавиши будут срабатывать по умолчанию при нажатии клавиш Enter или Esc

# Свойство формы AutoScroll

- ▶ Допустим, у вас получилась очень большая форма. Для прокрутки ее элементов включайте свойство:
- ▶ `AutoScroll=true` (по умолчанию его значение `false`)



AutoScaleMode	Font
<b>AutoScroll</b>	<b>True</b>
AutoScrollMargin	0; 0



# Свойство AutoSize

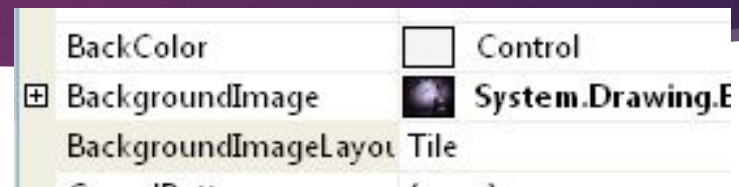
- ▶ Свойства AutoSize и AutoSizeMode позволяют установить размеры формы автоматически.

И вертикальный, и горизонтальный размеры формы будут меняться автоматически.

AutoSize	True
AutoSizeMode	GrowAndShrink

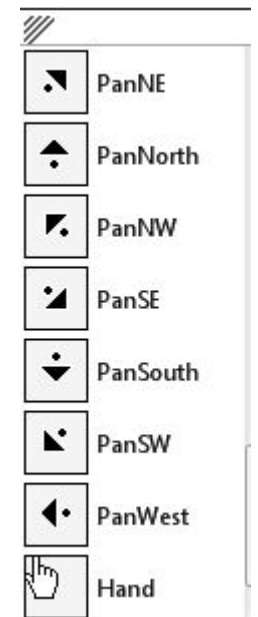
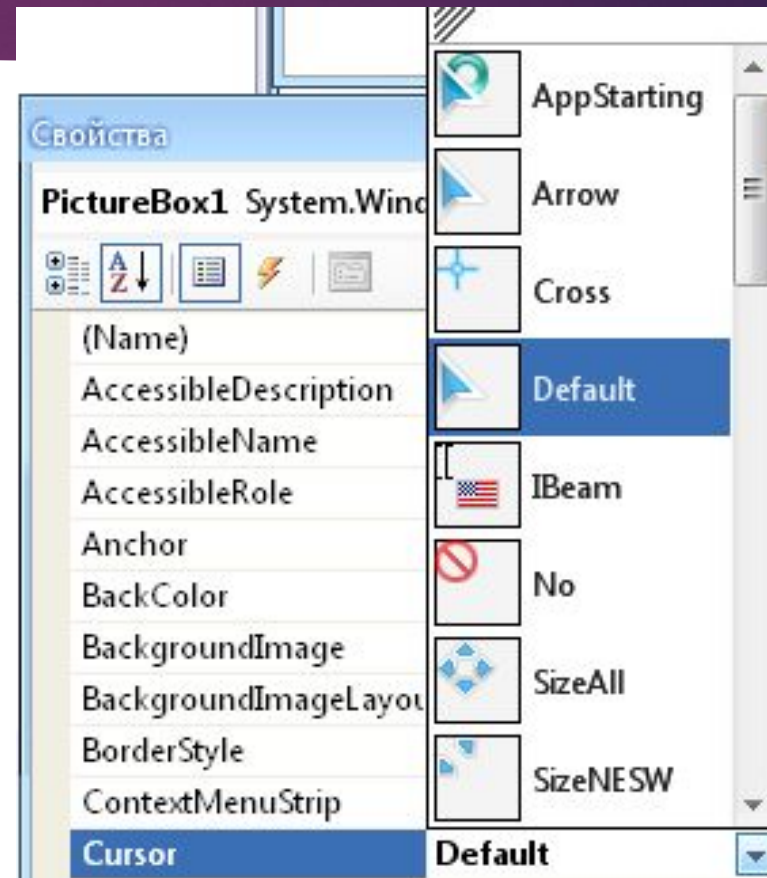
# Оформление фона

- ▶ BackColor
  - ▶ Заливка цветом
- ▶ BackgroundImage
  - ▶ Фоновый рисунок



# Свойство формы Cursor

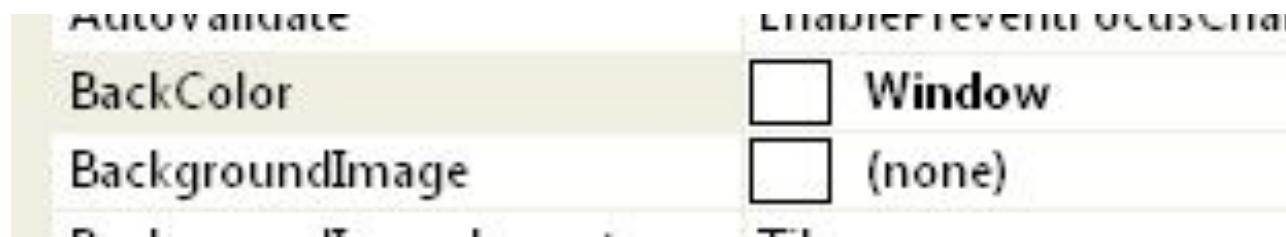
- задает вид курсора



# Стиль оформления формы

- ▶ `FormBorderStyle` позволяет изменить стиль оформления формы. Можно вообще убрать оформление формы (`none`).

Этот эффект особенно будет заметен, если еще подобрать соответствующий фон.



# Смена иконки

- ▶ В свойстве формы Icon укажите местоположение файла с новой иконкой (файл с расширением .ico)



# Свойства формы

- ▶ **MinimumSize** - При помощи этого свойства можно установить минимальные размеры формы, на которые ее можно растянуть
- ▶ **Opacity** - Чем меньше значение этого свойства, тем прозрачнее будет форма, да и вообще все окно. Короче это свойство задает прозрачность.
- ▶ **TransparencyKey** - Значение этого свойства задает цвет, который будет прозрачный. Например, сделай фон формы красный, и значение этого свойства - цвет красный. И форма будет прозрачной.
- ▶ **ShowInTaskbar** - отображается ли форма на панели задач, допустим вы хотите, чтобы ваше приложение после запуска было не видно и только иконка помещалась в трей, будет логичным установить это свойство в False.
- ▶ **WindowState** - состояние формы при запуске.
- ▶ Значения:
  - ▶ 0 - Normal; Обычное состояние формы.
  - ▶ 1 - Minimized; Названия говорят сами за себя. При запуске форма будет свернута.
  - ▶ 2 - Maximized. При запуске форма будет развернута во весь экран.

# Свойства объекта TextBox:

- ▶ **BackColor** - Фон текст. Поля
- ▶ **BorderStyle** - Внешний вид рамки объекта
- ▶ **CharacterCasing** - Свойство меняет регистр всех символов в текст. поле. Свойство может принимать 3 значения: `CharacterCasing.Lower` - прописные символы, `CharacterCasing.Normal` - регистр не меняется, `CharacterCasing.Upper` - заглавные символы
- ▶ **Font** - Свойства шрифта
- ▶ **ForeColor** - Цвет шрифта
- ▶ **MaxLeight** - Максимальная вместимость текстового поля, например, если значение этого свойства равно 5, то в текст. поле нельзя будет ввести больше 5 символов.
- ▶ **MultiLine** - если значение этого свойства равно `False`, то текстовое поле будет в однострочном режиме, если `True`, то в многострочном режиме.
- ▶ **Lines** - Используется в режиме проектирования, если `MultiLine = True`. Это свойство аналогично свойству **Text**, но в отличии от свойства **Text** в него можно вставлять текст состоящий из нескольких строк.

# Свойства объекта TextBox:

- ▶ **PasswordChar** - это свойство используется для защиты паролей, если значение этого свойства будет равно '\*', то введенные символы в текстовое поле будут шифроваться знаком '\*'.
- ▶ **ScrollBar** - это свойство нужно для установки полос прокрутки, может принимать 4 значения: **Horizontal** - горизонтальная полоса прокрутки, **Vertical** - вертикальная, **Both** - вертикальная и горизонтальная, **None** - без полос прокрутки. Свойство работает только если `MultiLine = True`.
- ▶ **ReadOnly** - Если значения свойства = `True`, то текст. поле доступно только для чтения



# Свойства объекта TextBox:

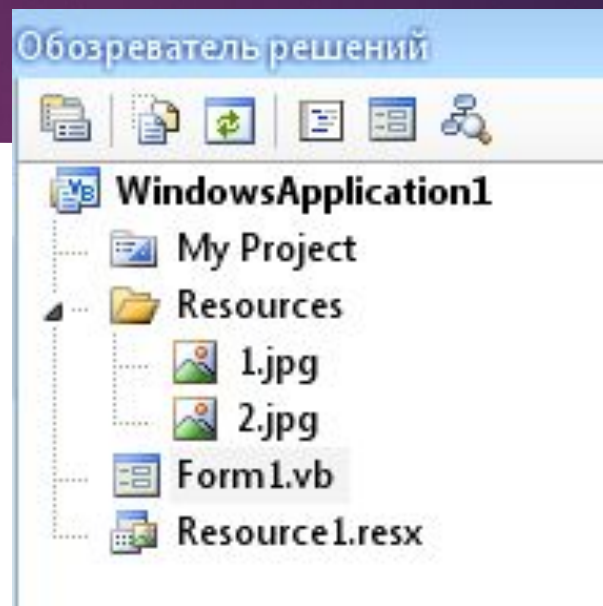
- ▶ **RightToLeft** - Если значение этого свойства = *RightToLeft.Yes*, то текст в текст. поле будет справа, если = *RightToLeft.no* то слева.
- ▶ **Text** - Текст, который будет в текст. поле. По умолчанию значение этого свойства - имя объекта.
- ▶ **TextAlign** - Выравнивание текста(по центру, по левому и правому краю)
- ▶ **Visible** - Видимость объекта, если значение = *True*, то объект видим, если *False*, то не видим.
- ▶ Location (x, y) - координаты левого верхнего угла
- ▶ Size (Width; Height) - размер (ширина; высота)

# Многооконный интерфейс

- ▶ При организации многооконного интерфейса можно использовать дополнительные свойства формы или объектов для того, чтобы определенные клавиши срабатывали по умолчанию или отображалась полоса прокрутки, или использовались индивидуальные элементы дизайна.

# Ресурсы проекта

- ▶ Конструктор ресурсов является средством пользовательского интерфейса, позволяющим управлять ресурсами (например, изображениями, значками, а также звуковыми и другими файлами) проекта.



# Спецификаторы `public`, `private`

- ▶ Спецификаторы доступа `private` и `public` управляют видимостью элементов класса. Элементы, описанные после служебного слова `private`, видимы только внутри класса. Этот вид доступа принят в классе по умолчанию. Интерфейс класса описывается после спецификатора `public`. Действие любого спецификатора распространяется до следующего спецификатора или до конца класса. Можно задавать несколько секций `private` и `public`, порядок их следования значения не имеет.

1. Переменные, описанные внутри метода, не будут видны за пределами этого метода. Например:

```
void MethodA()  
{  
    // Описываем переменную delta  
    int delta = 7;  
}
```

```
void MethodB()  
{  
    // Ошибка: переменная delta в этом методе неизвестна!  
    int gamma = delta + 1;  
}
```

Переменные, описанные внутри класса, являются *глобальными* и доступны для всех методов этого класса, например:

```
class Form1 : Form
{
    int a = 5;

    void Method()
    {
        // Переменная a здесь действительна
        MessageBox.Show(a.ToString());
    }
}
```

# Обработка событий

# Общие сведения о событиях (Windows Forms)

- ▶ Событие — это действие, на которое можно ответить (или которое можно обработать) с помощью кода. События возникают в результате действий пользователя, например при щелчке мышью или при нажатии клавиши, а также при выполнении программного кода или операций системы.
- ▶ Приложения, работающие на основе событий, в ответ на событие выполняют код. Для каждой формы и элемента управления предоставляется стандартный набор событий, которые могут быть запрограммированы. Если произойдет одно из этих событий, вызывается код, если он существует в соответствующем обработчике событий.
- ▶ Типы событий, вызываемых объектами, различны, но многие типы являются общими для большинства элементов управления. Например, большинство объектов будет обрабатывать событие Click. Если пользователь щелкает форму, выполняется код в обработчике событий Click формы.



# Создание обработчиков событий в Windows Forms

- ▶ Обработчик событий — это процедура в коде, определяющая действия, которые требуется выполнить при возникновении события, например когда пользователь нажимает кнопку или когда в очередь сообщений поступает очередное сообщение. При возникновении события выполняется обработчик (или обработчики) событий, который получает это событие. События могут быть назначены нескольким обработчикам, а методы, обрабатывающие отдельные события, могут динамически меняться. Можно также использовать конструктор Windows Forms Designer для создания обработчиков событий.

# Обработчик событий

- ▶ Обработчик событий — это метод, связанный с событием. При возникновении события выполняется код внутри обработчика событий. В каждом обработчике событий существует два параметра, которые позволяют правильно обработать событие.

```
private void button4_Click(object sender, EventArgs e)
{
    var result = music.Except(dancing);
    listBox3.Items.Clear();
    foreach (string word in result)
    {
        listBox3.Items.Add(word);
    }
}
```

# Сведения, передаваемые в событие

Когда происходит какое-либо событие (например, событие `Click` при нажатии на кнопку), в обработчик этого события передаются дополнительные сведения об этом событии в параметре `e`.

Например, при щелчке кнопки мыши на объекте возникает событие `MouseClick`. Для этого события параметр `e` содержит целый ряд переменных, которые позволяют узнать информацию о нажатии:

- `Button` – какая кнопка была нажата;
- `Clicks` – сколько раз была нажата и отпущена кнопка мыши;
- `Location` – координаты точки, на которую указывал курсор в момент нажатия, в виде объекта класса `Point`;
- `X` и `Y` – те же координаты в виде отдельных переменных.



# Написание программы обработки события

С каждым элементом управления на форме и с самой формой могут происходить события во время работы программы. Например, с кнопкой может произойти событие – нажатие кнопки, а с окном, которое проектируется с помощью формы, может произойти ряд событий: создание окна, изменение размера окна, щелчок мыши на окне и т. п. Эти события могут обрабатываться в программе. Для обработки таких событий необходимо создать обработчики события – специальный *метод*. Для создания обработчика события существует два способа.

Первый способ – создать обработчик для события по умолчанию

Поместите на форму кнопку, которая описывается элементом управления Button. С помощью окна свойств измените заголовок (Text) на слово «Привет» или другое по вашему желанию. Отрегулируйте положение и размер кнопки.

После этого два раза щелкните мышью на кнопке, появится текст программы:

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

}
```

Это и есть обработчики события нажатия кнопки. Вы можете добавлять свой код между скобками { }. Например, наберите:

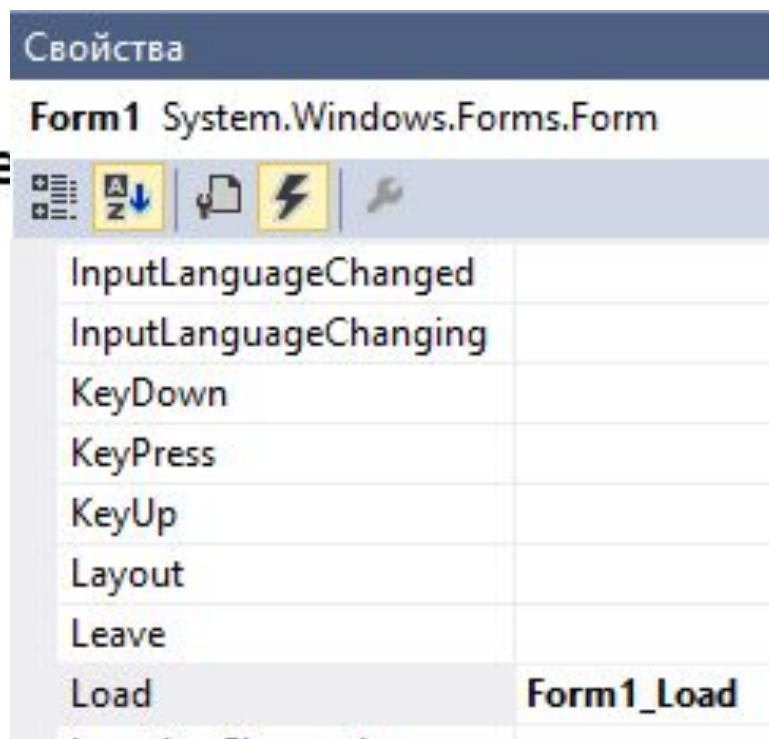
```
MessageBox.Show("Привет, " + textBox1.Text + "!");
```

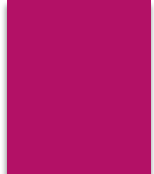


Второй способ создания обработчика события заключается в выборе соответствующего события для выделенного элемента на форме. При этом используется окно свойств и его закладка ⚡. Рассмотрим этот способ. Выделите форму щелчком по ней, чтобы вокруг нее появилась рамка из точек. В окне свойств найдите событие Load. Щелкните по данной строчке дважды мышкой. Появится метод:

```
private void Form1_Load(object sender  
{
```

11

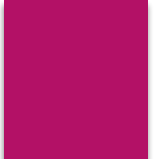




Каждый элемент управления имеет свой набор обработчиков событий, однако некоторые из них присущи большинству элементов управления. Наиболее часто применяемые события описаны ниже:

- `Activated`: форма получает это событие при активации.
- `Load`: возникает при загрузке формы. В обработчике данного события следует задавать действия, которые должны происходить в момент создания формы, например установка начальных значений.
- `KeyPress`: возникает при нажатии кнопки на клавиатуре. Параметр `e.KeyChar` имеет тип `char` и содержит код нажатой клавиши (клавиша `Enter` клавиатуры имеет код `#13`, клавиша `Esc` — `#27` и т. д.). Обычно это событие используется в том случае, когда необходима реакция на нажатие одной из клавиш.



- 
- `KeyDown`: возникает при нажатии клавиши на клавиатуре. Обработчик этого события получает информацию о нажатой клавише и состоянии клавиш `Shift`, `Alt` и `Ctrl`, а также о нажатой кнопке мыши. Информация о клавише передается параметром `e.KeyCode`, который представляет собой перечисление `Keys` с кодами всех клавиш, а информацию о клавишах-модификаторах `Shift` и др. можно узнать из параметра `e.Modifiers`.
  - `KeyUp`: является парным событием для `KeyDown` и возникает при отпускании ранее нажатой клавиши.
  - `Click`: возникает при нажатии кнопки мыши в области элемента управления.
  - `DoubleClick`: возникает при двойном нажатии кнопки мыши в области элемента управления.