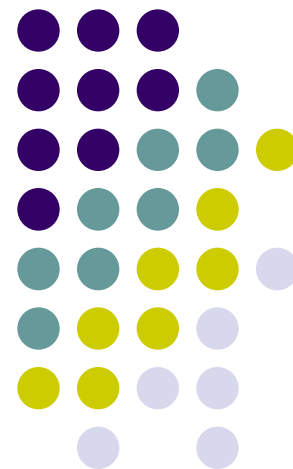
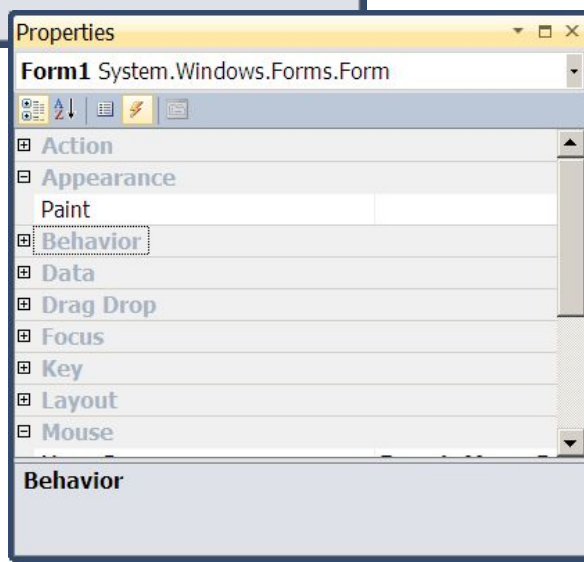
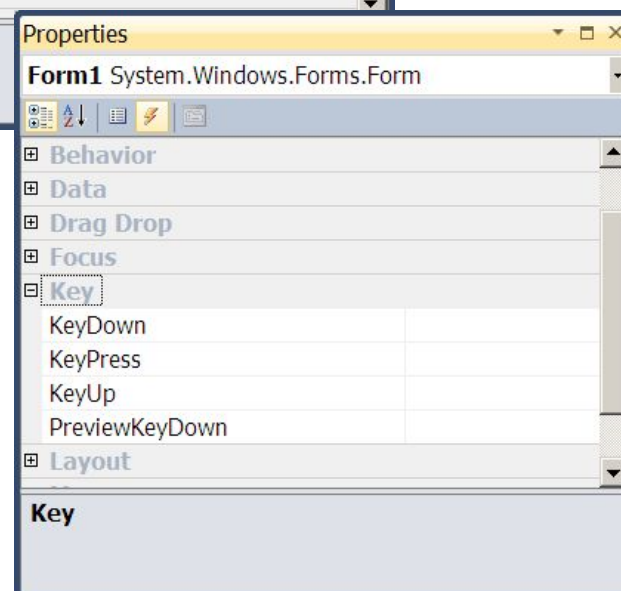
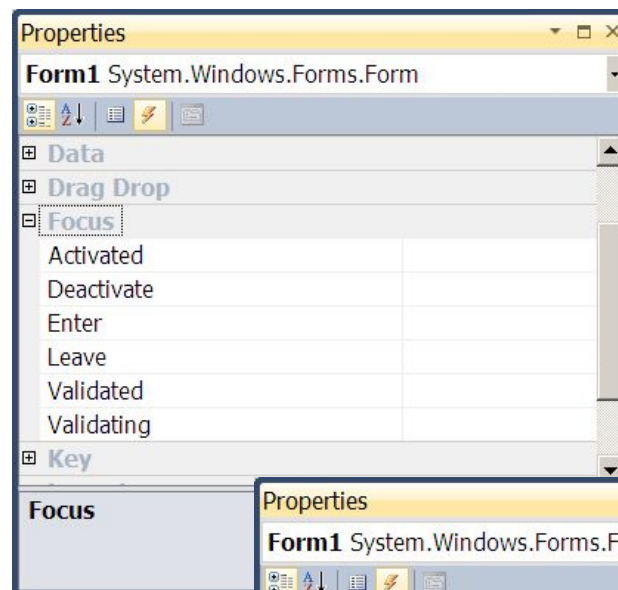
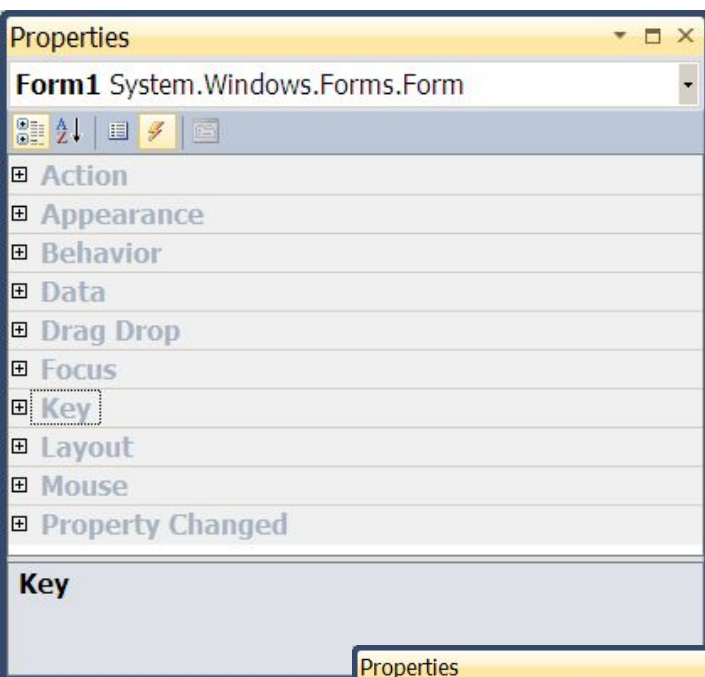


# События



В общем случае концепции событий при программировании приложения среды выполнения Windows подобны моделям событий в самых популярных языках программирования.

## Виды событий элементов управления





Структура обработчика события элементов управления:

```
private void ИмяЭл_Событие(object sender, EventArgs e)
```

Обработчик, созданный вами для события, может обращаться к двум значениям, которые доступны как вводные при каждом вызове обработчика. Первое значение — это *sender*, представляющий собой ссылку на объект, к которому прикреплен обработчик. Параметр *sender* типизирован как базовый тип **Object**. Часто используется такой прием, как преобразование *sender* в тип с большей точностью. Этот прием полезен, если предполагаются проверки или изменения состояния самого объекта *sender*. Исходя из проекта приложения вы выбираете тип, в который можно безопасно преобразовать *sender*, учитывая участок прикрепления обработчика или другую специфику проекта.

Второе значение — это данные события, которые обычно включаются в определения синтаксиса как параметр *e*. Изучив параметр *e* делегата, который сопоставлен определенному обрабатываемому событию, можно выяснить, какие свойства доступны для данных события. Для некоторых событий значения определенных свойств данных события не менее важны, чем сам факт возникновения события. EventArgs класс служит базовым классом для всех классов, представляющих данные события.



```
private void button3_KeyPress(object sender,  
    KeyPressEventArgs e)
```

```
private void Form1_MouseDown(object sender,  
    MouseEventArgs e)
```

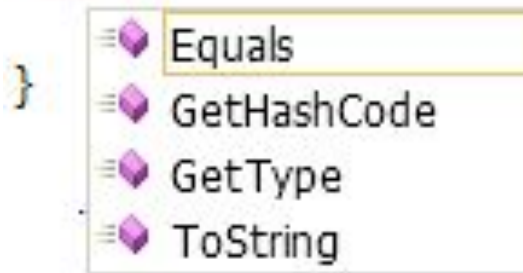
```
private void Form1_Paint(object sender, PaintEventArgs  
    e)
```

```
private void Form1_Enter(object sender, EventArgs e)
```

# EventArgs



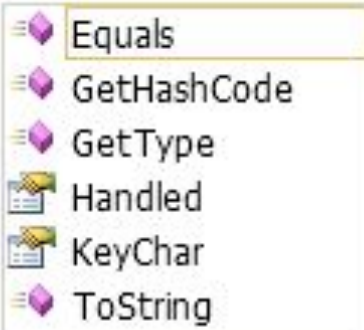
```
private void Form1_Enter(object sender, EventArgs e)
{e.
```





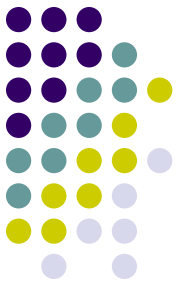
# KeyPressEventArgs

```
private void button3_KeyPress(object sender, KeyPressEventArgs e)
{e.
```

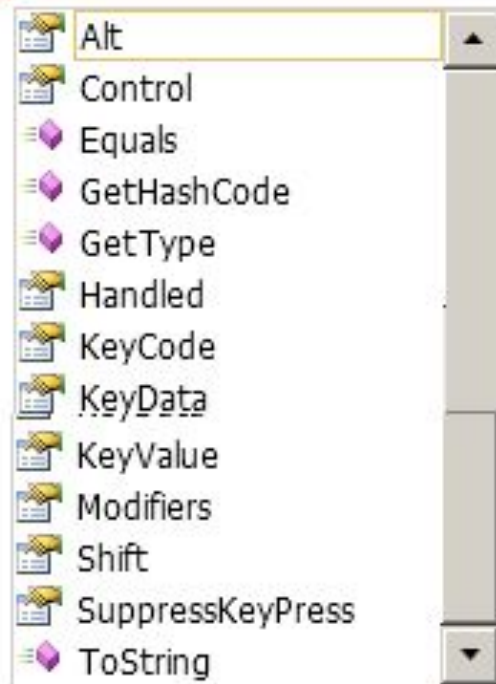


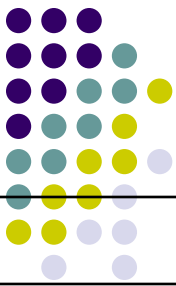
Имя	Описание
<a href="#">Handled</a>	Возвращает или задает значение, указывающее, является ли <a href="#">KeyPress</a> событие было обработано. Установка <a href="#">Handled</a> В true отмена KeyPress событие. Это позволяет сохранять элемент управления из обработки ключевое нагрузку.
<a href="#">KeyChar</a>	Возвращает или задает символ, соответствующий отжтому

# EventArgs



```
private void Form1_KeyDown(object sender, EventArgs e)
{
    e.
```



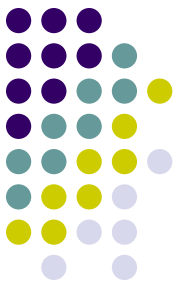
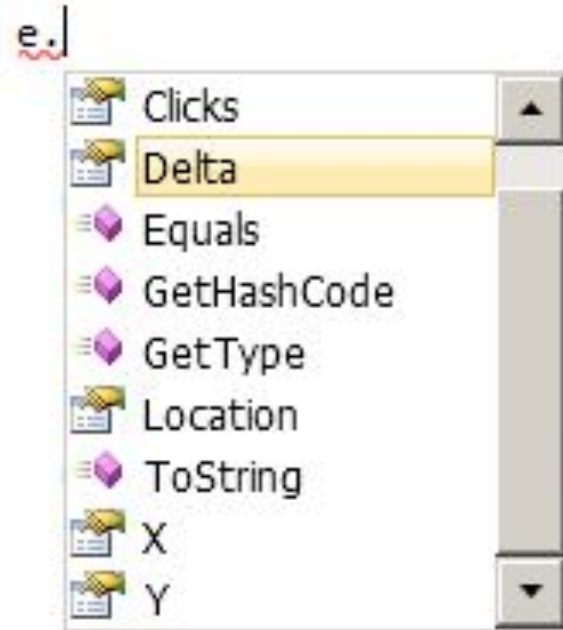


Имя	Описание
<a href="#">Alt</a>	Получает значение, показывающее, была ли нажата клавиша ALT.
<a href="#">Control</a>	Получает значение, показывающее, была ли нажата клавиша CTRL.
<a href="#">Handled</a>	Возвращает или задает значение, указывающее, было ли обработано событие.
<a href="#">KeyCode</a>	Получает код клавиатуры для события <a href="#">KeyDown</a> или события <a href="#">KeyUp</a> .
<a href="#">KeyData</a>	Получает данные, касающиеся клавиши, для события <a href="#">KeyDown</a> или <a href="#">KeyUp</a> .
<a href="#">KeyValue</a>	Получает значение клавиатуры для события <a href="#">KeyDown</a> или <a href="#">KeyUp</a> .
<a href="#">Modifiers</a>	Получает флаги модификаторов для события <a href="#">KeyDown</a> или события <a href="#">KeyUp</a> . Флаги указывают, сочетание CTRL, ключи МИГРАЦИИ и ALT отжал.
<a href="#">Shift</a>	Получает значение, показывающее, была ли нажата клавиша SHIFT.
<a href="#">SuppressKeyPre</a>	Возвращает или задает значение, указывающее, должно ли событие передаться клиенту и базовому элементу.

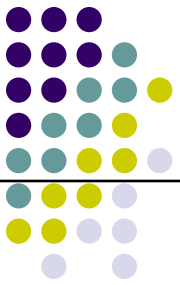


# MouseEventArgs

```
private void Form1_MouseDown(object sender, MouseEventArgs e)  
{
```



# MouseEventArgs

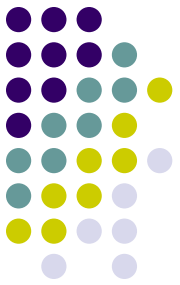


Имя	Описание
<a href="#">Button</a>	Возвращает, какая кнопка мыши была нажата.
<a href="#">Clicks</a>	Возвращает количество раз была нажата и была отпущена кнопка мыши.
<a href="#">Delta</a>	Возвращает число со знаком, указывающее количество делений, на которое повернулось колесико мыши, умноженное на константу WHEEL_DELTA. Делением называется один зубец колесика мыши.
<a href="#">Location</a>	Получает расположение указателя мыши в момент создания события мыши.
<a href="#">X</a>	Получает координату x мыши в момент создания события мыши.
<a href="#">Y</a>	Получает координату указателя мыши в момент

# Событие Paint

Происходит при перерисовке элемента управления.

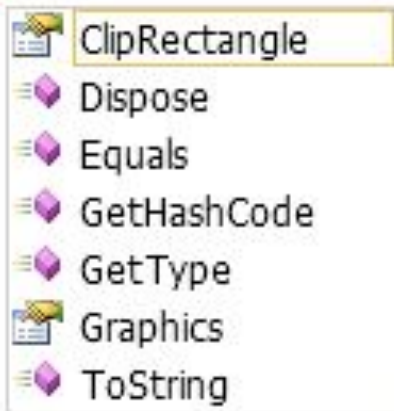
## PaintEventArgs



```
private void Form1_Paint(object sender, PaintEventArgs e)
```

```
{ e.
```

```
}
```



Имя	Описание
<a href="#">ClipRectangle</a>	Возвращает прямоугольник, в котором можно рисовать.
<a href="#">Graphics</a>	Возвращает графику, используемый для рисования.

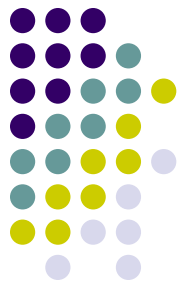


# Control – класс

Определяет базовый класс для элементов управления, являющихся компонентами с визуальным представлением.

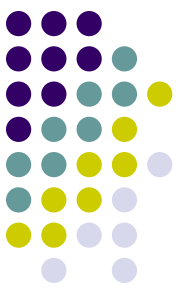
<u>Location</u>	Получает или задает координаты левого верхнего угла элемента управления относительно левого верхнего угла контейнера.
<u>MouseButton</u>	Получает значение, показывающее, какая из кнопок мыши нажата в данный момент.
<u>MousePosition</u>	Получает позицию указателя мыши в экранных координатах.
<u>Name</u>	Возвращает или задает имя элемента управления.

```
private void textBox2_KeyPress(object sender,  
KeyPressEventArgs e)
```



```
{ bool zpt=false;  
    if (char.IsDigit(e.KeyChar) == true) return;  
    if (e.KeyChar == (char)Keys.Back) return;  
    if (textBox2.Text.IndexOf(',') != -1)  
        zpt = true;  
    if (zpt == true) { e.Handled = true; return; }  
    if (e.KeyChar == ',') return;  
    e.Handled = true;  
}
```

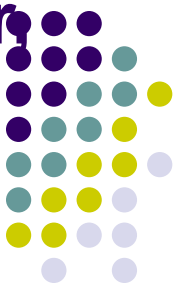
```
private void textBox1_KeyDown(object sender,  
EventArgs e)
```



```
{ int i;  
    i=Convert.ToInt32(textBox1.Text);  
    if (e.KeyData == Keys.Down) i--;  
    textBox1.Text = i.ToString();  
}
```

```
private void button1_MouseDown(object sender,  
MouseEventArgs e)
```

```
{    if (e.Button == MouseButton.Right)  
        button1.BackColor = Color.Coral;  
    if (e.Button == MouseButton.Left)  
        button1.BackColor = Color.Gray;  
}
```



# Добавление и удаление обработчиков событий в программный код

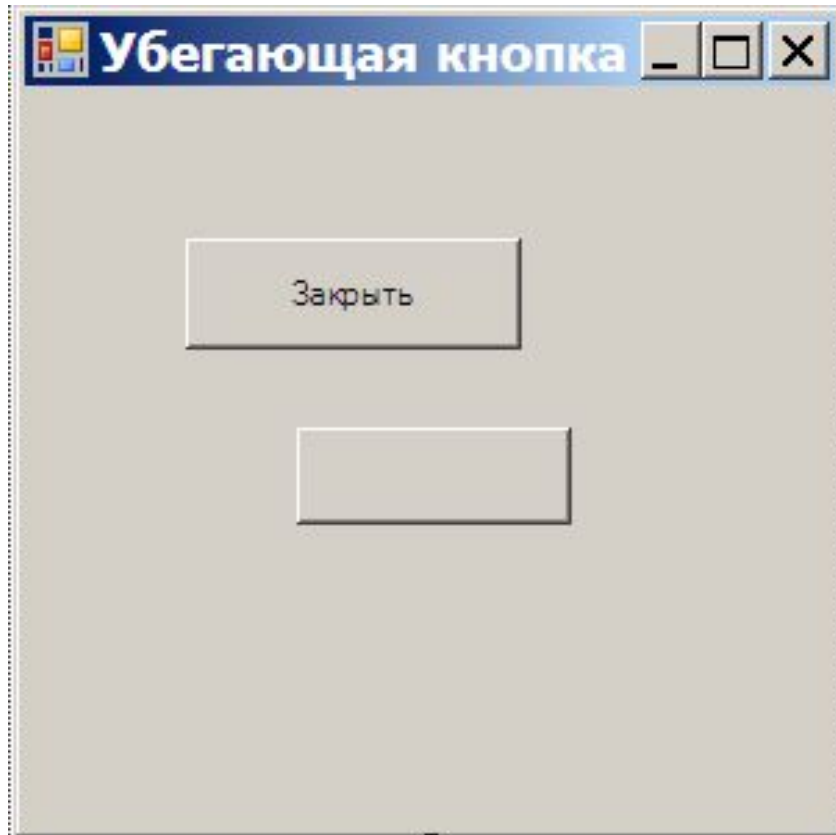


В C# синтаксис предусматривает использование оператора `+=`. Для регистрации обработчика справа от оператора добавляется ссылка на имя метода обработчика событий.

Для удаления



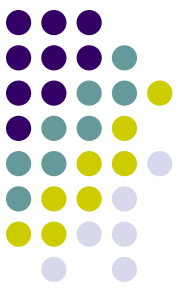
# Проект «Убегающая кнопка»



```
public partial class Form1 : Form
```

```
{  
    private Random r = new Random();  
    public Form1()  
    {  
        InitializeComponent();  
    }  
    private void button1_Click(object sender, EventArgs  
e)
```

```
{  
    Close();  
  
}
```



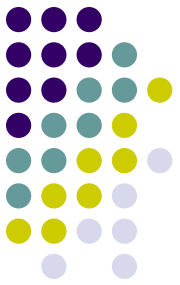


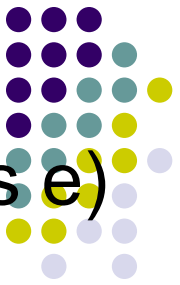
```
private void button2_MouseMove(object sender,  
    MouseEventArgs e)  
{  
    if (Control.ModifierKeys == Keys.Control) return;  
    button2.Location = new  
        Point(r.Next(ClientRectangle.Width - 5),  
            r.Next(ClientRectangle.Height - 5));  
}
```

# Подключаемый обработчик

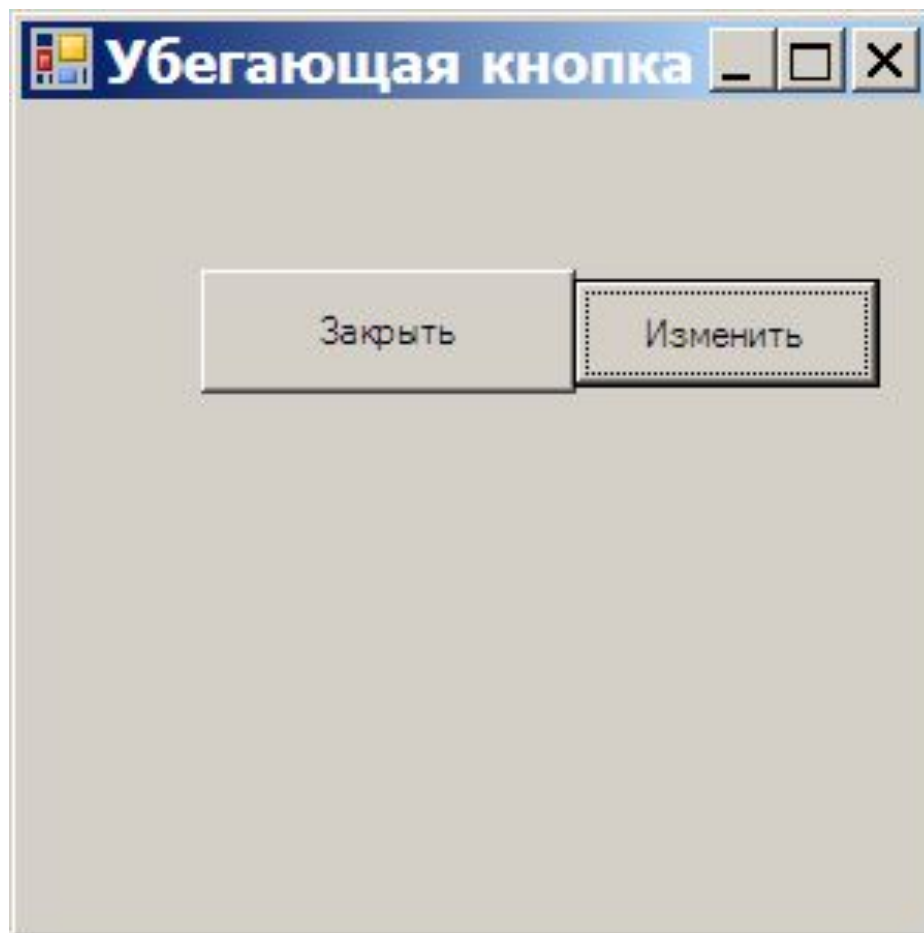
```
private void button2_Click2(object sender, EventArgs e)
{
    if (WindowState == FormWindowState.Normal)
    {
        WindowState = FormWindowState.Maximized;
    }

    else
    {
        WindowState = FormWindowState.Normal;
    }
}
```





```
private void button2_Click(object sender, EventArgs e)
{
    button2.Text = "Изменить";
    button2.MouseMove -= button2_MouseMove;
    button2.Click -= button2_Click;
    button2.Click += button2_Click2;
}
```





```
private void Form1_MouseDown(object sender,
    MouseEventArgs e)
{
    button1.Location = new Point(e.X - button1.Width / 2,
        e.Y - button1.Height / 2);
    if (button2.Text != "")
    {
        button2.Text = "";
        button2.MouseMove += button2_MouseMove;
        button2.Click += button2_Click;
        button2.Click -= button2_Click2;
    }
}
```

