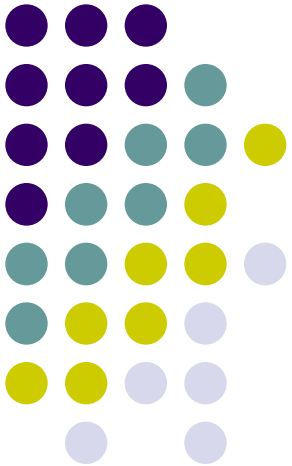
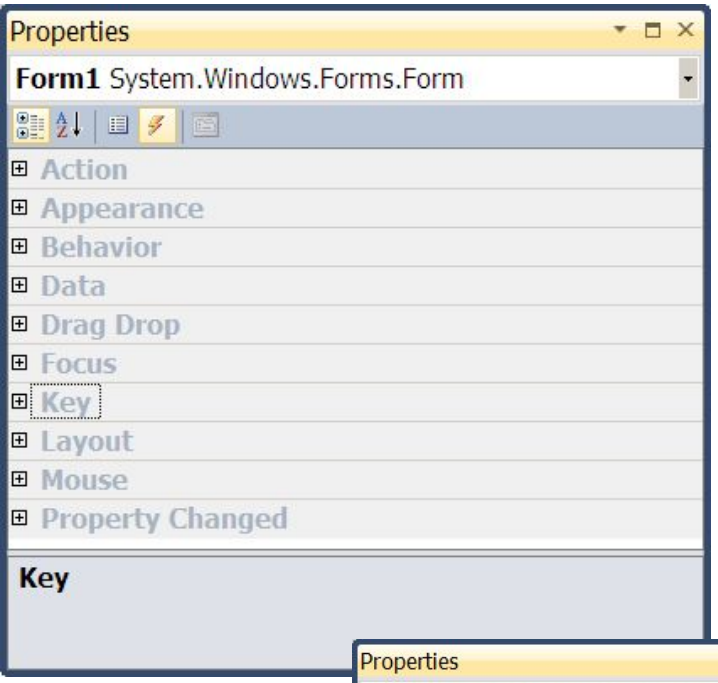
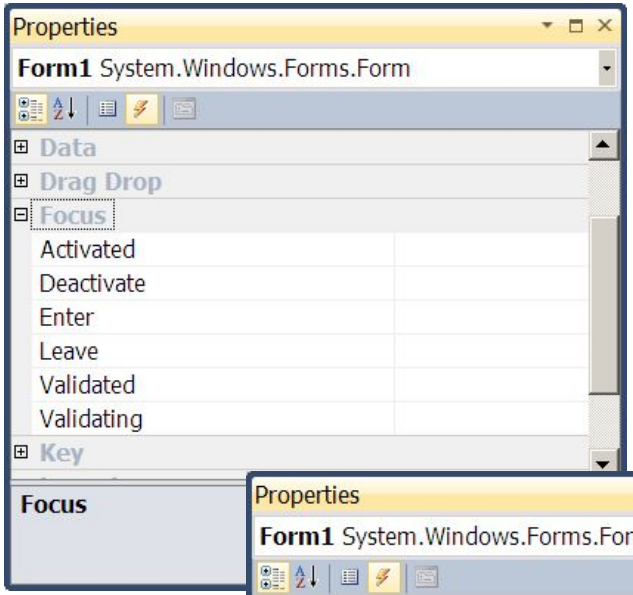
События

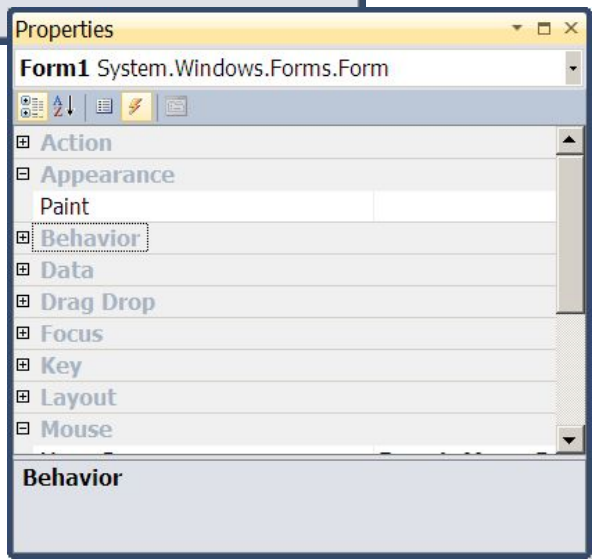


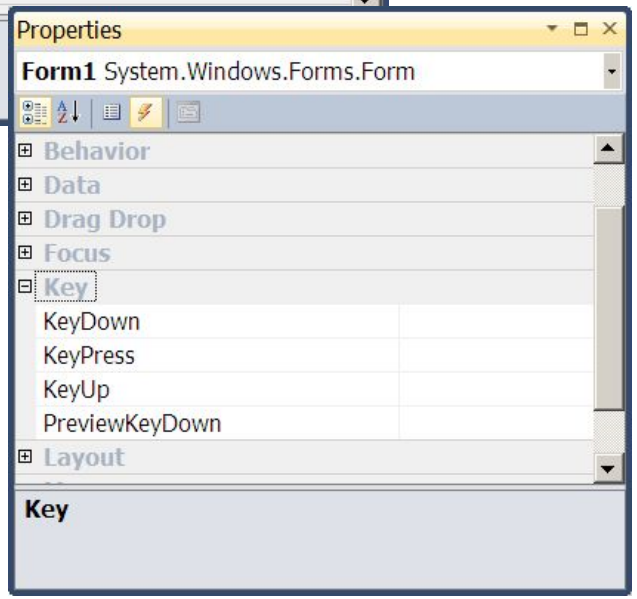
В общем случае концепции событий при программировании приложения среды выполнения Windows подобны моделям событий в самых популярных языках программирования.

Виды событий элементов управления







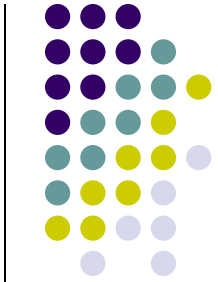


Структура обработчика события элементов управления:

private void ИмяЭл_Событие(object sender, EventArgs e)

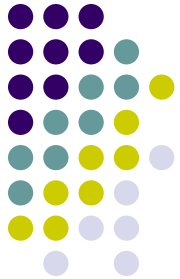
Обработчик, созданный вами для события, может обращаться к двум значениям, которые доступны как вводные при каждом вызове обработчика. Первое значение — это *sender*, представляющий собой ссылку на объект, к которому прикреплен обработчик. Параметр *sender* типизирован как базовый тип **Object**. Часто 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)
```

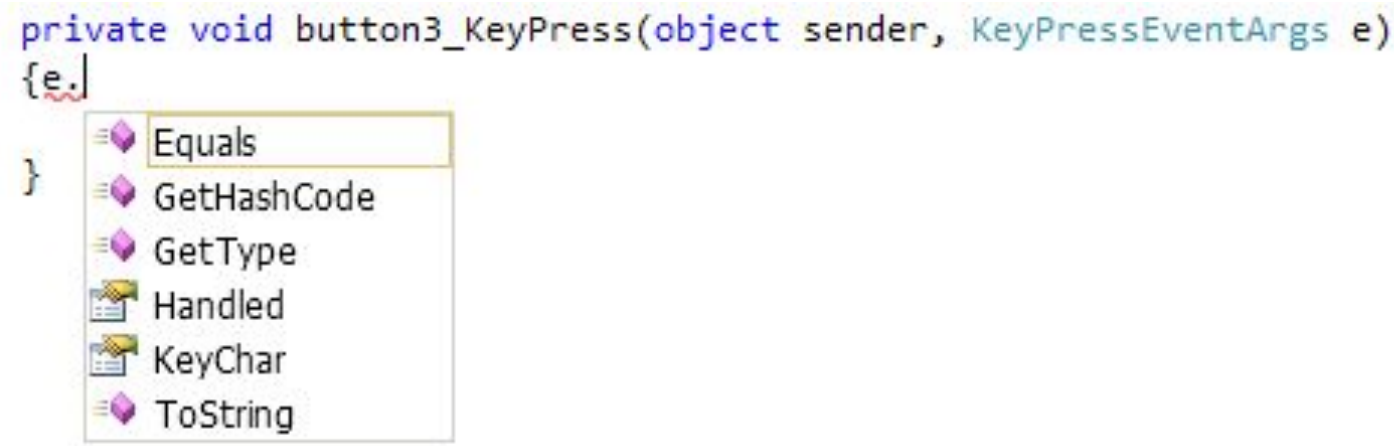
EventArgs



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

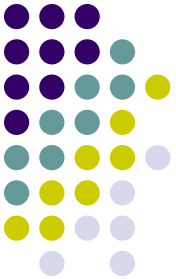
- Equals
- GetHashCode
- GetType
- ToString

KeyPressEventArgs



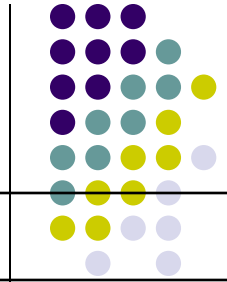
Имяя	Описание
Handled	Возвращает или задает значение, указывающее, является ли KeyPressВозвращает или задает значение, указывающее, является ли KeyPress событие было обработано. Установка Handled В true отмена KeyPress событие. Это позволяет сохранять элемент управления из обработки ключевое нагрузку.
KeyChar	Возвращает или задает символ, соответствующий отжатому

KeyEventArgs



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

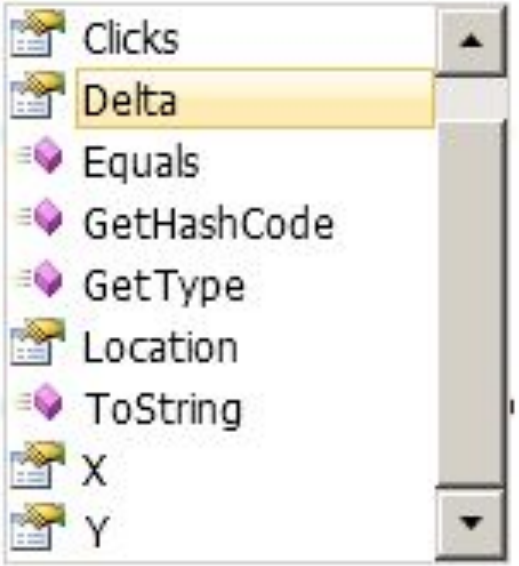
- Alt
- Control
- Equals
- GetHashCode
- GetType
- Handled
- KeyCode
- KeyData
- KeyValue
- Modifiers
- Shift
- SuppressKeyPress
- ToString



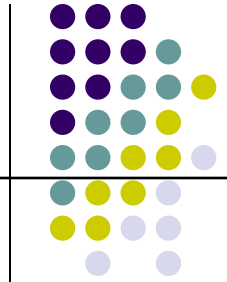
Имя	Описание
Alt	Получает значение, показывающее, была ли нажата клавиша ALT.
Control	Получает значение, показывающее, была ли нажата клавиша CTRL.
Handled	Возвращает или задает значение, указывающее, было ли обработано событие.
KeyCode	Получает код клавиатуры для события KeyDown или события KeyUp.
KeyData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

MouseEventArgs

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



MouseEventArgs



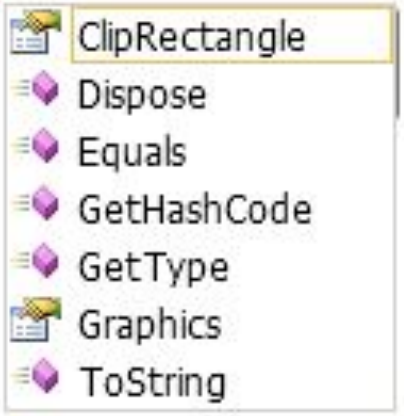
Имя	Описание
Button	Возвращает, какая кнопка мыши была нажата.
Clicks	Возвращает количество раз была нажата и была отпущена кнопка мыши.
Delta	Возвращает число со знаком, указывающее количество делений, на которое повернулось колесико мыши, умноженное на константу WHEEL_DELTA. Делением называется один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

Событие Paint

Происходит при перерисовке элемента управления.

PaintEventArgs

```
private void Form1_Paint(object sender, PaintEventArgs e)
{e.
}
```



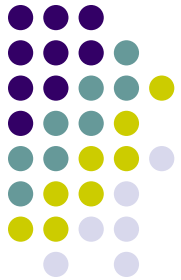
Имя	Описание
ClipRectangle	Возвращает прямоугольник, в котором можно рисовать.
Graphics	Возвращает графику, используемый для рисования.

Control – класс

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

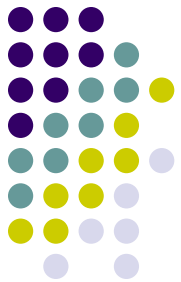
Location	Получает или задает координаты левого верхнего угла элемента управления относительно левого верхнего угла контейнера.
MouseButtons	Получает значение, показывающее, какая из кнопок мыши нажата в данный момент.
MousePosition	Получает позицию указателя мыши в экранных координатах.
Name	Возвращает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

private void textBox2_KeyPress(object sender, KeyPressEventArgs e)



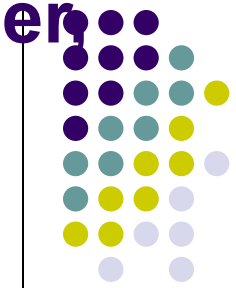
```
{  bool zpt=false;
        if (char.IsDigit(e.KeyChar) == true) return;
        if (e.KeyChar == (char)Keys.Back) return;
        if (textBox2.Text.IndexOf(',') != -1)
            zpt = true;
        if (zpt == true) { e.Handled = true; return; }
        if (e.KeyChar == ',') return;
        e.Handled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

private void textBox1_KeyDown(object sender, KeyEventArgs e)

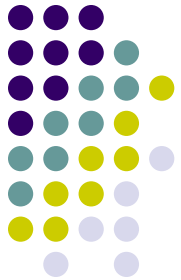


```
{ int i;
        i=Convert.ToInt32(textBox1.Text);
        if (e.KeyData == Keys.Down) i--;
        textBox1.Text = i.ToString();
    }
```

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

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



Добавление и удаление обработчиков событий в программный код

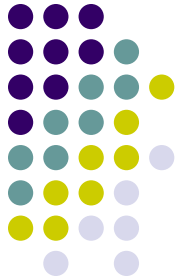


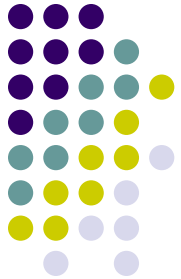
В C# синтаксис предусматривает использование оператора +=. Для регистрации обработчика справа от оператора добавляется ссылка на имя метода обработчика событий.

Для удаления

Проект «Убегающая кнопка»

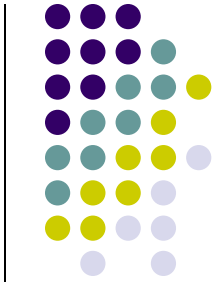
```
public partial class Form1 : Form
    {
        private Random r = new Random();
        public Form1()
        {
            InitializeComponent();
        }
        private void button1_Click(object sender, EventArgs
    e)
        {
            Close();

        }
```





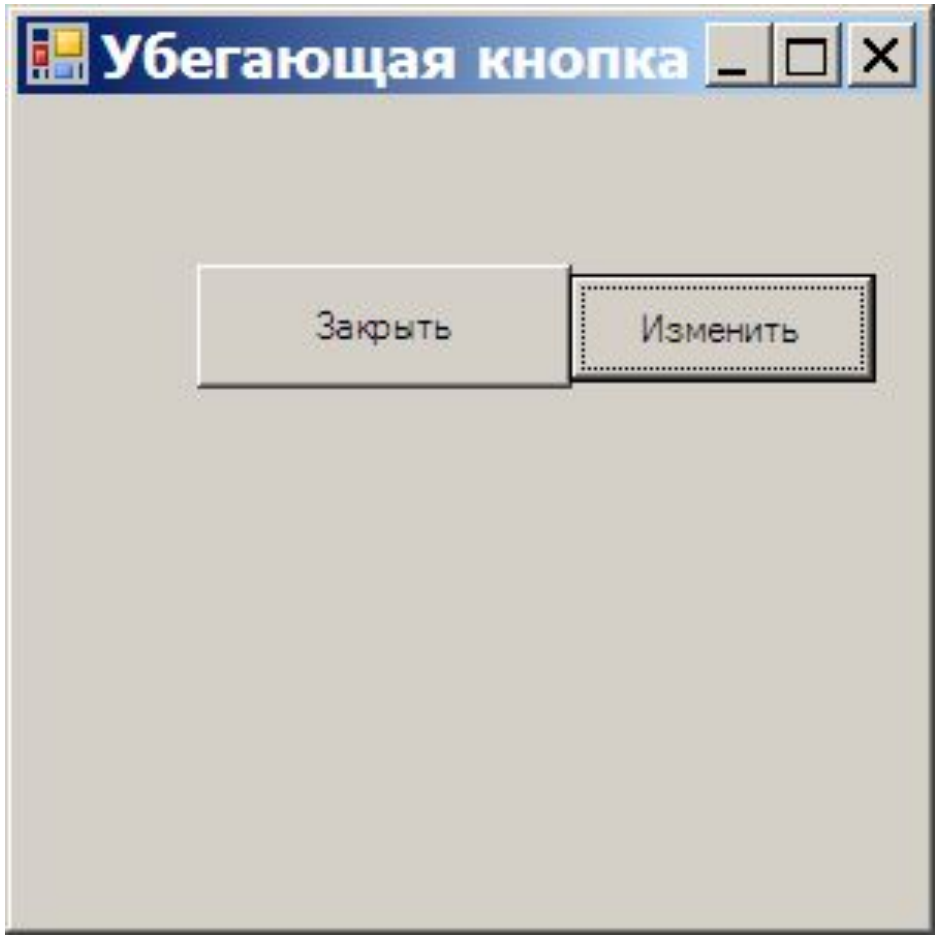
```
private void button2_MouseMove(object sender,
   MouseEventArgs e)
{
   if (Control.ModifierKeys == Keys.Control) return;
   button2.Location = new
   Point(r.Next(ClientRectangle.Width - 5),
         r.Next(ClientRectangle.Height - 5));
}
```

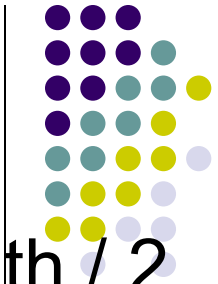
Подключаемый обработчик



```
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;
        }
```


Убегающая кнопка
Закрыть
Изменить



```
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;
          }
}
```


Убегающая кнопка
Закрыть