

Лекция 14 СОБЫТИЯ

14.1 Понятие события

В основе работы операционной системы Windows лежит принцип событийного управления. Это означает, что и сама система и все приложения, написанные для Windows, после запуска ожидают действий пользователя или сообщений операционной системы и реагируют на них определенным образом – через очередь сообщений приложения формируются события, на которые приложение должно реагировать.

Каждый элемент управления (кнопка или строка меню) имеет свой идентификатор. Когда Вы нажимаете на кнопку или выбираете строку меню, в очередь сообщений приложения **Windows** заносит сообщение, содержащее идентификатор использованного элемента управления. Таким образом операционная система **Windows** направляет сообщение от использованного элемента управления в очередь того приложения, к которому принадлежит данный элемент управления.

Если в окне формы была нажата кнопка, то факт нажатия кнопки (сообщение) через операционную систему **Windows** вернется в очередь сообщений окна нашей формы и у кнопки возникнет событие **Click**.

Итак, механизм появления события понятен, но что это событие?

Из лекции 7 (состав класса) мы определили события класса как специальные методы, позволяющие классу реагировать на действия пользователя или на определенные изменения в программе.

Из этого определения событие и метод – обработчик события являются синонимами.

Уточним эти понятия. Каждый объект является экземпляром некоторого класса. Свойства класса определяют его состояние, а методы класса определяют его поведение.

У всех объектов один и тот же набор свойств, но значения свойств объектов различны, так что объекты одного класса могут находиться в разных состояниях. Например, значение свойства «доход», у объектов класса «Служащий», может очень сильно различаться, и эти объекты могут находиться в разных состояниях. Например, «Хочу купить автомобиль, но не имею возможности ...».

Все объекты обладают одними и теми же методами и набор событий у всех объектов одного класса один и тот же, но вот методы, обрабатывающие возникшие события, могут быть разные. Например, у двух кнопок окна формы («Ввод данных», «Переход на форму 2») одинаковые события **Click**, но обработчики событий (их коды) разные.

Таким образом, событие как свойство класса, являясь единым для всех объектов, имеет различную реализацию для каждого конкретного экземпляра класса.

События позволяют задать индивидуальное поведение объекта в специфических ситуациях, когда возникает некоторое сообщение.

Таким образом, события класса это специальные методы, позволяющие классу реагировать на действия пользователя или на определенные изменения в программе.

Для многих действий пользователя (они предсказуемы) разработаны заготовки для обработчиков событий – смотри Properties ->Events.

14.2 Некоторые часто используемые события среды Visual Studio.NET

Для каждого элемента управления **Windows**-приложений определен свой набор событий, на которые он может реагировать. Заготовка шаблона обработчика события формируется двойным щелчком на поле, расположенное справа от имени соответствующего события на странице Events окна Properties для выбранного элемента управления.

Наиболее часто используемыми событиями для элементов управления являются следующие:

Click, DoubleClick – одинарный или двойной щелчок мышки;

KeyDown, KeyUp – нажатие и отпускание любой клавиши и их сочетаний;

KeyPress – нажатие клавиши, имеющей ASCII-код

MouseDown, MouseUp – нажатие и отпускание кнопки мыши;

MouseMove – перемещение мыши;

Paint – возникает при необходимости прорисовки формы.

Например, если дважды «Кликнуть» на кнопку, расположенную в окне нашей формы, то будет сформирован обработчик этого события со следующим заголовком:

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

Если нам необходимо обрабатывать событие, связанное с двойным нажатием кнопки мыши в пределах формы, то в «событиях» формы необходимо дважды «Кликнуть» на поле, расположенное справа от события KeyDown при этом появится шаблон обработчика этого события со следующим заголовком: `private void Form1_MouseDoubleClick(object sender, MouseEventArgs e).`

В качестве примеров привожу шаблоны еще двух обработчиков событий:

```
private void textBox1_KeyDown(object sender, KeyEventArgs e)
private void Form1_MouseClick(object sender, MouseEventArgs e)
```

Попробуйте определить сами, когда они формируются?

Анализируя шаблоны обработчиков событий можно отметить общую закономерность – у всех шаблонов два формальных параметра, причем первый (`object sender`) у всех одинаковый.

Во многих учебниках по программированию на языке C# при пояснении механизма работы события используется модель «публикация – подписка», в которой один класс, являющийся отправителем сообщения (`sender`), публикует сообщение, а другие классы, являющиеся получателями сообщения (`receivers`), подписываются на получение этих сообщений.

В соответствии с этой терминологией первый формальный параметр всех обработчиков сообщений `sender` типа `object` (класс `object` является базовым любого класса языка C#) является отправителем сообщения (иногда говорят – источником сообщения).

Второй формальный параметр шаблонов обработчиков событий является переменная типа объект точнее ссылкой на объект класса `XXXEventArgs`, который содержит связанные с событием параметры (фактически сообщение), например, для обработчиков мышки можно получить координаты `e.X` `e.Y` курсора мышки. Класс `XXXEventArgs` (`XXX` имя события) во всех стандартных событиях является наследником системного класса `EventArgs`.

При описании модели «публикация – подписка» отмечалось, что получатели сообщений (`receivers`), подписываются на получение этих сообщений. То есть для элементов управления и формы необходимо указывать (подписывать) сообщения, на которые должен реагировать этот элемент. Поскольку все методы обработчиков событий имеют единый формат записи, то их можно объединять (подписывать) с помощью многоадресных делегатов – делегатов, способных указывать на любое количество функций. Для каждого элемента в файле `Form1.Designer.cs`, в разделе инициализации помимо определения свойств элементов управления, определяются перечень событий, на которые должен реагировать этот элемент управления. Например, фрагмент инициализации формы с определением перечня событий, на которые она должна реагировать, имеет следующий вид:

```
private void InitializeComponent()
{
    . . .
    this.MouseDoubleClick += new
        System.Windows.Forms.MouseEventHandler
            (this.Form1_MouseDoubleClick);

    this.Paint += new
        System.Windows.Forms.PaintEventHandler(this.Form1_Paint);
    this.MouseClick += new System.Windows.Forms.MouseEventHandler
        (this.Form1_MouseClick);
    this.MouseMove += new System.Windows.Forms.MouseEventHandler
        (this.Form1_MouseMove);
    . . .
}
```

Необходимо отметить, что все визуальные элементы управления произошли от класса `Control`, в котором представлены наиболее важные для работы события.

14.3 Пример использования стандартных событий классов

Рассмотрим чисто учебную программу, использующие некоторые «стандартные» события классов.

В программе использован обработчик события одинарного нажатия левой кнопки мышки на форме, при этом из всех запусков обработчика программа реагирует только на нажатие кнопки мышки, когда ее курсор находится в левом верхнем углу формы:

```
private void Form1_MouseClick(object sender, MouseEventArgs e)
{
    if ((e.X < 20 && e.X > 0 && e.Y < 20 & e.Y > 0) && p == 0)
```

Этот прием часто используется программистами для формирования реакции программы на «клик» мышки в определенных местах экрана, например, карты города или фото группы студентов – легко написать необходимые комментарии при «попадании» мышки в определенный участок формы.

Обработчик события перемещения мышки по форме (дополнительное условие $p==1$) позволяет записать положения курсора мышки в массив 100 точек.

Просмотр массива в графическом режиме осуществляется с помощью обработчика двойного нажатия левой кнопки мышки.

С помощью обработчика – нажатия клавиши в окне редактора осуществляется начальная установка программы.

Исходный код программы:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        public int p,i,j;
        int[,] tk = new int[100, 2];
        public Form1()
        {
            InitializeComponent();
            textBox1.Text = "";
            p = 0; i = 0; j = 0;
        }
        private void Form1_MouseClick(object sender, MouseEventArgs e)
        {
            if ((e.X < 20 && e.X > 0 && e.Y < 20 & e.Y > 0) && p == 0)
            {
                textBox1.AppendText("Вы кликнули мышкой в верхнем левом углу
                                     \r\n");

                p++;
            }
        }
        private void Form1_MouseMove(object sender, MouseEventArgs e)
```

```

{
    if (p == 1)
        if (i < 100) { tk[i, 0] = e.X; tk[i, 1] = e.Y; i++; }
        else
        {
            textBox1.AppendText("Заполнение массива закончено \r\n");
            p++;
        }
    }
    private void Form1_MouseDoubleClick(object sender,
                                         MouseEventArgs e)
    {
        this.Invalidate();
    }
    private void Form1_Paint(object sender, PaintEventArgs e)
    {
        Pen myPen = new Pen(Color.Red, 2);
        Graphics g = e.Graphics;
        for (int n = 0; n < 100; n++)
            g.DrawEllipse(myPen, tk[n, 0], tk[n, 1], 2, 2);
    }
    private void textBox1_KeyDown(object sender, KeyEventArgs e)
    {
        p = 0; i = 0;
    }
}
}

```

Работа программы:

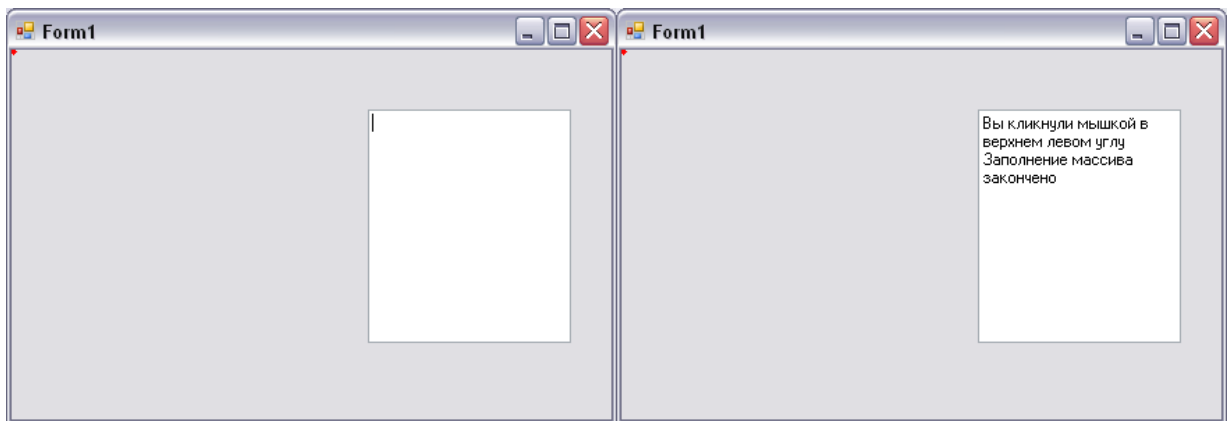


Рисунок 14.1 – Запуск программы и заполнение массива

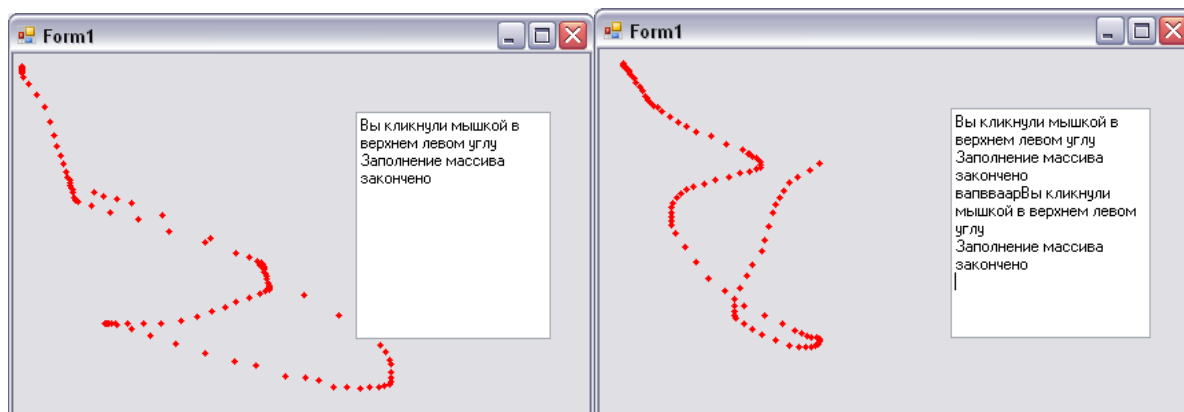


Рисунок 14.2 – Просмотр массива, сброс программы и ее повторный запуск

14.4 Нестандартные события классов

Помимо стандартных событий – как реакция программы на действия пользователя, события могут формироваться и внутри **Windows**-приложений с помощью специальных методов.

События, формируемые самим приложением, позволяют организовать динамическую (во время работы приложения) связь между различными объектами, например, продажа некоторого товара в магазине должно автоматически изменять содержимое многих документов – общий доход от продажи, документ по продаже конкретного товара, наличие этого товара на складе и рекомендации по пополнению товара и т.д.

Таким образом, возникает необходимость информирования некоторых объектов приложения о событии, происшедшем в объекте источнике сообщения.

Механизм взаимодействия источника события с его получателями (иногда их называют клиентами) основан на использовании делегата. В приложении объявляется экземпляр делегата, которому соответствуют стандартные методы обработчиков событий.

Далее необходимо определить класс, являющегося источником события (sender), и в нем метод описывающий событие, и метод иницирующий событие.

В процессе работы приложения объекты обработчиков событий (клиентов), которые хотят получать уведомление об изменении состояния источника, необходимо включить в список методов, обрабатываемых делегатом. Этот процесс называется регистрацией обработчиков событий..

При появлении события все зарегистрированные методы поочередно, с помощью делегата, вызываются на выполнение.

Работу механизма взаимодействия источника события с его получателями рассмотрим на следующем учебном примере.

Задача 14.1 Сформировать массив из 10 случайных целых чисел в диапазоне от минус 5 до 10. **Предположим, что отрицательные значения недопустимы и для них мы будем формировать события.** Необходимо

напечатать массив, вычислить его сумму и нарисовать график изменения значений.

Если значение массива число отрицательное, то необходимо формировать событие, не которое должны реагировать два обработчика. Первый должен поменять знак у отрицательного числа, а второй должен изменить сумму чисел в соответствии с новым значением элемента массива. Для наглядности предусмотреть вывод графика измененных значений массива.

Итак, для создания и использования события нужно, прежде всего, объявить делегата, с помощью которого реализуется связь между источником события и его клиентами.

В библиотеке .NET описано огромное количество стандартных делегатов, предназначенных для реализации механизма обработки событий. Большинство этих классов оформлено по одним и тем же правилам:

- имя делегата включает название события и заканчивается суффиксом **EventHandler**;

- делегат имеет два формальных параметра. Первый параметр задает источник события и имеет тип **object**. Второй параметр задает аргументы события и имеет тип **EventArgs** или производный от него.

Рекомендуется при объявлении делегата придерживаться этих правил, например:

```
public delegate void ZamenaEventHandler(object sender, ZamenaEventArgs  
arge);
```

В этом примере мы создаем делегата для обработки события **Zamena** (замена), связанного с изменениями, которые должны произойти в объекте-источнике события. В описании делегата указаны два аргумента: объект **sender**, создавший событие, и объект **arge** типа **ZamenaEventArgs**, содержащий связанные с событием параметры.

Поскольку нам понадобится передавать только индекс массива, то лучше определить класс **ZamenaEventArgs** следующим образом:

```
public class ZamenaEventArgs : EventArgs  
{  
    public int item;  
}
```

Следующим этапом в реализации механизма обработки событий является определение класса, являющегося источником события (**sender**), и в нем метода, инициирующего событие. Этот метод имеет специальный формат записи и во многом определяется форматом записи соответствующего ему делегата – после определения спецификатора доступа к методу (обычно это **public**) записывается служебное слово **event**, после которого указывается тип, заданный делегатом, и имя события. Например:

```
public event ZamenaEventHandler Zamena;
```

Полное описание класса выглядит следующим образом:

```
class sobit
```

```

{
    public event ZamenaEventHandler Zamena;
    public void prov(ZamenaEventArgs arge)
    {
        if (masi[arge.item] < 0)
        {
            Zamena(this,arge);
        }
    }
}

```

На следующем этапе реализации механизма обработки событий необходимо определиться с классами получателями событий. Особенность этих классов является то, что в них должны находиться методы обработчиков событий, формат записи которых должен совпадать с форматом записи соответствующего делегата. Например:

```

class zam1
{
    public void OnZam1(object sender, ZamenaEventArgs e)
    {
        masi[e.item] = masi[e.item] * (-1);
    }
}

```

Регистрацию обработчиков событий необходимо производить для объектов (не классов) классов обработчиков событий. Это возможно только во время работы приложения – объекты (переменные) создаются во время работы приложения, например, во время инициализации формы:

```

public Form1()
{
    InitializeComponent();
    zam1 z1 = new zam1();
    zam2 z2 = new zam2();
    so.Zamena += z1.OnZam1;
    so.Zamena += z2.OnZam2;
}

```

Для демонстрации работы приложения использованы две кнопки – «Формирование массива» и «Проверка массива и запуск событий».

Исходный код программы имеет следующий вид:

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

```

```

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        public delegate void ZamenaEventHandler(object sender,
                                                ZamenaEventArgs arge);

        public int cob=0;
        public static int sum = 0;
        public static int[] masi = new int[10];
        public class ZamenaEventArgs : EventArgs
        {
            public int item;
        }
        class sobit
        {
            public event ZamenaEventHandler Zamena;
            public void prov(ZamenaEventArgs arge)
            {
                if (masi[arge.item] < 0)
                {
                    Zamena(this, arge);
                }
            }
        }
        sobit so = new sobit();
        class zam1
        {
            public void OnZam1(object sender, ZamenaEventArgs e)
            {
                masi[e.item] = masi[e.item] * (-1);
            }
        }
        class zam2
        {
            public void OnZam2(object sender, ZamenaEventArgs e)
            {
                //if(masi[e.item]>0)//Проверка очередности обработки событий
                sum = sum + 2 * masi[e.item];
            }
        }
        public Form1()
        {
            InitializeComponent();
            zam1 z1 = new zam1();
            zam2 z2 = new zam2();
            so.Zamena += z1.OnZam1;
            so.Zamena += z2.OnZam2;
        }
        private void button1_Click(object sender, EventArgs e)
        {
            cob = 0; sum = 0;
            string ss="";

```

```

Random rnd = new Random();
for(int i=0;i<10;i++)
{
    masi[i] = rnd.Next() % 15 - 5;
    sum = sum + masi[i];
    ss = ss + masi[i].ToString() + " ";
}
textBox1.AppendText("Исходный массив: \r\n");
textBox1.AppendText(ss + "\r\n");
textBox1.AppendText("Сумма элементов="+sum.ToString()+"\r\n");
this.Invalidate();
}
private void button2_Click(object sender, EventArgs e)
{
    cob=1;
    ZamenaEventArgs zz = new ZamenaEventArgs();
    for (int i = 0; i < 10; i++)
    {
        zz.item = i;
        so.prov(zz);
    }
    textBox1.AppendText("Сумма элементов="+sum.ToString()+"\r\n");
    this.Invalidate();
}
private void Form1_Paint(object sender, PaintEventArgs e)
{
    Graphics g = e.Graphics;
    g.DrawLine(new Pen(Brushes.Blue, 2), 10, 200, 250, 200);
    Pen myPen = new Pen(Color.Red, 2);
    if (cob == 0)
    {
        for (int i = 1; i < 10; i++)
            g.DrawLine(myPen,i*10, (200-masi[i-1]*10), (i+1)*10, (200-
masi[i]*10));
    }
    if (cob == 1)
    {
        for (int i = 1; i < 10; i++)
            g.DrawLine(myPen,i*10+110, (200-masi[i-1]*10), (i+1)*10+110,
(200-masi[i]*10));
    }
}
}
}

```

Работа программы:

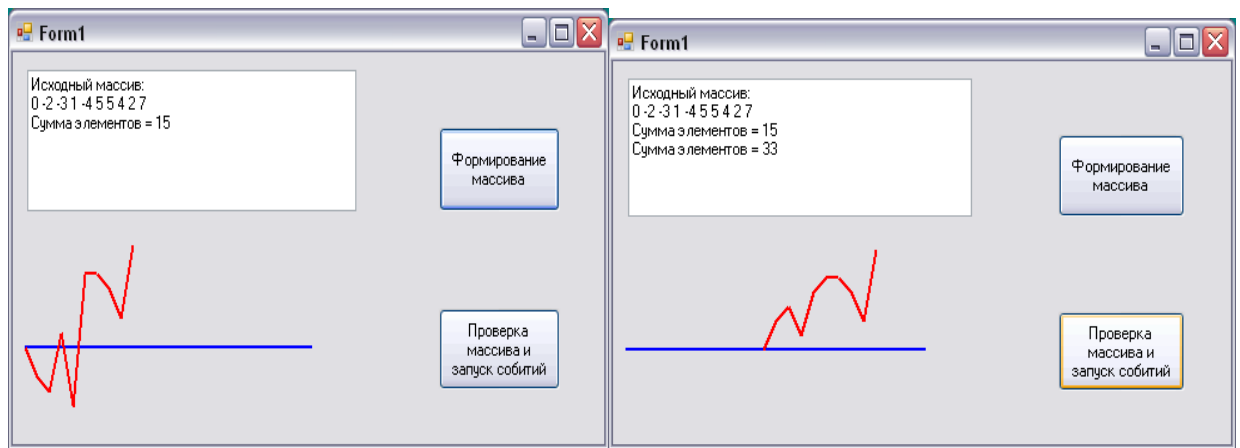


Рисунок 14.3 Работа приложения с обработкой событий