

Лекция 2 ЭЛЕМЕНТЫ УПРАВЛЕНИЯ

2.1 Технология визуального проектирования форм

Среда визуального программирования C# .NET включает в себя Windows.Forms.Designer (конструктор или дизайнер форм Windows) – инструмент, позволяющий в интерактивном режиме выполнять визуальное проектирование формы, размещая на ней необходимые элементы управления. Преимущества Windows.Forms.Designer в том, что Вы можете размещать элементы управления на форме в соответствии с Вашим представлением красоты и при этом не думать о конкретных значениях многих свойств этих элементов, например, о свойствах (точнее числовых значениях этих свойств), определяющих их местоположение или размер. Значения большинства свойств задаются автоматически конструктором формы. Однако конструктор формы может помочь Вам до определенного момента, а потом Вам придется писать код программы вручную, попутно разбираясь в том, что для Вас сгенерировал Windows.Forms.Designer.

В Windows приложении (в отличие от консольного приложения) конструктором формы автоматически создаются несколько классов с расширением .cs, например, класс с именем Form1.cs и класс с именем Program.cs.

Классы в C# синтаксически не являются неделимыми и могут состоять из нескольких частей, каждая из которых начинается с ключевого слова “partial”(частичный). Таковым является и построенный автоматически класс Form1. Возможность разбиения описания одного класса на части облегчает работу над большим классом. Каждая часть класса хранится в отдельном файле со своим именем. Например, для примера предыдущей лекции, автоматически были созданы два файла Form1 с расширением .cs – Form1.cs и Form1.Designer.cs.

Первая часть класса Form1, хранящаяся в файле "Form1.cs", предназначена для разработчика – именно в ней располагаются автоматически создаваемые обработчики событий, происходящих с элементами управления, код которых создается самим разработчиком. Такая технология программирования, основанная на работе с формами, называется визуальной, событийно управляемой технологией программирования.

Например, фрагмент файла Form1.cs, рассмотренного на предыдущей лекции:

```
namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
    }
}
```

```
private void button1_Click(object sender, EventArgs e)
{
    int a,b,c,p;
    a = Convert.ToInt32(textBox1.Text);
    b = Convert.ToInt32(textBox2.Text);
    c = Convert.ToInt32(textBox3.Text);
    p = a + b + c;
    ...
}
```

Вторая часть класса Form1 находится в файле с именем "Form1.Designer.cs". Эта часть класса заполняется автоматически конструктором формы. Когда мы занимаемся визуальным проектированием формы и размещаем на ней различные элементы управления, меняем их свойства, придаем форме нужный вид, задаем обработчиков событий для элементов управления, то конструктор формы транслирует наши действия в действия над объектами соответствующих классов, создает соответствующий код и вставляет его в нужное место класса Form1.

Мы предполагаем (надемся), что Вы не должны вмешиваться в работу конструктора формы и корректировать эту часть кода класса Form1. Тем не менее, иметь представление о его работе или даже понимать код, созданный конструктором формы иногда очень полезно.

Ниже приведен фрагмент файла Form1.Designer.cs, рассмотренного на предыдущей лекции:

```
namespace WindowsFormsApplication1
{
    partial class Form1
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        ...
        private void InitializeComponent()
        {
            this.label1 = new System.Windows.Forms.Label();
            this.label2 = new System.Windows.Forms.Label();
            this.label3 = new System.Windows.Forms.Label();
            this.label4 = new System.Windows.Forms.Label();
            this.button1 = new System.Windows.Forms.Button();

            ...
            // label1
            //
            this.label1.AutoSize = true;
            this.label1.Location = new System.Drawing.Point(12,
9);

            this.label1.Name = "label1";
            this.label1.Size = new System.Drawing.Size(174, 13);
            this.label1.TabIndex = 0;
            this.label1.Text = "Введите сторону треугольника A";
            ... и т.д. всего на 3 страницах.
        }
    }
}
```

Класс Program.cs , автоматически создаваемый для нашего проекта, содержит статический метод Main(). При запуске нашей программы система Windows ищет метод Main() и начинает выполнять указания, стоящие в нем. Часто метод Main() называют точкой входа в программу.

Ниже приведен файл Program.cs, примера, рассмотренного на предыдущей лекции:

```
namespace WindowsFormsApplication1
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}
```

В отличие от консольных приложений, где тело метода Main() изначально было пустым и должно было заполняться разработчиком проекта, в Windows приложениях метод Main() уже заполнен необходимыми указаниями и, как правило, разработчиком не изменяется. Что же делает автоматически созданный метод Main()? Он работает с классом Application библиотеки FCL, вызывая поочередно три статических метода этого класса.

Метод Application.EnableVisualStyles(); представляет компоненты создаваемого приложения в стиле Windows XP.

Метод Application.SetCompatibleTextRenderingDefault(false);

Говорят, что его назначение понятно из текста.

Метод Application.Run(new Form1()); основной метод класса и обычно он единственный в Main().

Основную работу выполняет метод Run – в процессе его вызова создается объект класса Form1 и открывается форма - визуальный образ объекта, с которой может работать пользователь проекта. Если, как положено, форма спроектирована и заполнена элементами управления, то пользователю остается вводить собственные данные в поля формы и нажимать на кнопки. В ответ на возникающие события начинают работать обработчики событий, что приводит к желаемым (или не желанным) результатам работы программы. В следующем пункте лекции необходимо кратко рассмотреть назначение элементов управления, расположенных на панели **Toolbox**.

2.2 Элементы управления панели Toolbox

Элементы управления, или компоненты, помещают на форму из элементов управления Toolbox (View ► Toolbox). В этом разделе лекции кратко описаны простейшие элементы управления панели Toolbox.

При изучении элемента управления рекомендуется следующая последовательность действий. С помощью мышки переместите элемент на форме, выделите его, нажмите клавишу F1 и перейдите по ссылке ...overview (обзор). Изучите разделы Remarks и Example, затем перейдите по ссылке ...Members (элементы класса). Попробуйте получить представление о возможностях изучаемого класса, выделив главные из его свойств и открытых методов. После этого можно вернуться к заготовке приложения и начать экспериментировать со свойствами, а затем — с методами класса.

Изучение элементов управления начнем с элемента, который практически всегда присутствует на форме – текст комментариев.

2.2.1 Label – метка

Метка предназначена для размещения текста на форме. Размещаемый текст хранится в свойстве Text. Можно задавать шрифт текста (свойство Font), цвет фона (свойство BackColor), цвет шрифта текста (ForeColor) и выравнивание (свойство TextAlign) текста метки. Метка может автоматически изменять размер в зависимости от длины текста (свойство AutoSize = True). Можно разместить на метке изображение (свойство Image) и задать прозрачность (установить для свойства BackColor значение Color.Transparent). В этом случае будут видны компоненты, расположенные на форме за надписью.

Метка, как самостоятельный элемент управления, не может получать фокус ввода – «запоминать» положение курсора мышки и создавать обработчики событий на нажатие клавиш мышки, клавиатуры или других элементов управления.

2.2.2 Button – кнопка

Элемент управления Button может получать фокус ввода, при этом основное событие, обрабатываемое кнопкой, — щелчок мышью (Click). Кроме того, кнопка может реагировать на множество других событий — нажатие клавиш на клавиатуре и мыши, изменение параметров и т. д.

Если занести имя кнопки в свойство Accept Button формы, на которой расположена кнопка, то нажатие клавиши Enter вызывает событие Click, даже если кнопка не имеет фокуса ввода. Такая кнопка имеет дополнительную рамку и называется кнопкой по умолчанию.

Аналогично, если занести имя кнопки в свойство Cancel Button формы, на которой расположена кнопка, то нажатие клавиши Esc вызывает событие Click для этой кнопки.

Можно изменить начертание и размер шрифта текста кнопки, который хранится в свойстве `Text`, задать цвет фона и фоновое изображение так же, как и для метки.

Кнопка может содержать помимо надписи еще и изображение (свойство `Image` или `ImageList` вместе с `ImageIndex`).

Кнопки часто используются в диалоговых окнах. Как видно из названия, такое окно предназначено для диалога с пользователем и запрашивает у него какие-либо сведения (например, какой выбрать режим работы или какой файл открыть). Диалоговое окно обладает свойством модальности. Это означает, что дальнейшие действия с приложением невозможны до того момента, пока это окно не будет закрыто. Закрыть окно можно, либо подтвердив введенную в него информацию щелчком на кнопке `OK` (или `Yes`), либо отменив ее с помощью кнопки закрытия окна или, например, кнопки `Cancel`. Для сохранения информации о том, как было закрыто окно, у кнопки определяют свойство `DialogResult`. Это свойство может принимать стандартные значения из перечисления `DialogResult`, определенного в пространстве имен `System.Windows.Forms`. Значения перечисления приведены в таблице 2.1.

Таблица 2.1. Значения перечисления `DialogResult`

Значение	Описание	Значение	Описание
<code>None</code>	Окно не закрывается	<code>Ignore</code>	Нажата кнопка <code>Ignore</code>
<code>OK</code>	Нажата кнопка <code>OK</code>	<code>Yes</code>	Нажата кнопка <code>Yes</code>
<code>Cancel</code>	Нажата кнопка <code>Cancel</code>	<code>No</code>	Нажата кнопка <code>No</code>
<code>Abort</code>	Нажата кнопка <code>Abort</code>	<code>Retry</code>	Нажата кнопка <code>Retry</code>

2.2.3 Поле ввода `TextBox`

Компонент `TextBox` позволяет пользователю вводить и редактировать текст, который запоминается в свойстве `Text`. Можно вводить строки практически неограниченной длины (приблизительно до 32 000 символов), корректировать их, а также вводить защищенный текст (пароль) путем установки маски, отображаемой вместо вводимых символов (свойство `PasswordChar`). В однострочном режиме высота компонента автоматически меняется так, чтобы показывать только одну строку.

Свойство `Text` используется для ввода единственной строки, а свойство `Lines` — для ввода нескольких. Строки в этом свойстве хранятся в виде массива, что позволяет организовать индексный доступ к ним.

Для обеспечения возможности ввода и вывода нескольких строк устанавливают свойства `Multiline`, `ScrollBars` и `WordWrap`. Доступ только для чтения устанавливается с помощью свойства `ReadOnly`.

Элемент содержит методы очистки (`Clear`), выделения (`Select`), копирования в буфер (`Copy`), вставки из него (`Paste`) и другие. Может

обрабатывать множество событий, основными из которых являются KeyPress и KeyDown.

2.2.4 ListBox — список

Компонент ListBox представляет собой список с возможностью выбора одного или нескольких пунктов. Свойство SelectMode может иметь одно из следующих значений: None — выбор пунктов запрещен; One — можно выбирать только один пункт; MultiSimple — можно выбирать несколько пунктов; MultiExtended — можно выбирать несколько пунктов с учетом нажатых клавиш Shift и Ctrl: если нажата и удерживается клавиша Shift, выбирается непрерывный диапазон пунктов; если нажата и удерживается клавиша Ctrl, выбирается произвольный (необязательно непрерывный) диапазон пунктов. Если в свойство MultiColumn помещено значение True, пункты списка могут располагаться в несколько колонок, при этом свойство ColumnWidth определяет ширину колонок. Если колонки выйдут за ширину компонента, автоматически вставляется горизонтальная полоса прокрутки.

2.2.5 Переключатель RadioButton

Переключатель позволяет пользователю выбрать один из нескольких предложенных вариантов, поэтому переключатели обычно объединяют в группы. Если один из них устанавливается (свойство Checked), остальные автоматически сбрасываются. Программист может менять стиль и цвет текста, связанного с переключателем, и его выравнивание. Для переключателя можно задать цвет фона и фоновое изображение так же, как и для метки. Переключатели можно поместить непосредственно на форму, в этом случае все они составят одну группу. Если на форме требуется отобразить несколько групп переключателей, их размещают внутри компонента Group или Panel. Свойство Appearance управляет отображением переключателя: либо в традиционном виде (Normal), либо в виде кнопки (Button), которая «залипает» при щелчке на ней мышью.

2.3 Пример использования элементов управления

Для этого раздела выбран чисто учебный пример для вычисления тригонометрических функций синус, косинус и тангенс. Значение переменной (угол в радианах) задается в режиме диалога с программой. Также в режиме диалога задается имя вычисляемой функции и количество разрядов формата вывода функции на экран монитора — точность вычисления. Для реализации этой задачи в проекте использованы следующие элементы управления: Label, Button, Panel, RadioButton, ListBox и TextBox.

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 Form1()
        {
            InitializeComponent();
        }
        private void button1_Click(object sender, EventArgs e)
        {
            //Значения по умолчанию
            string ff = "F3";
            string fu = "sin";
            Double x=0;
            //Ввод значения угла
            x = Convert.ToDouble(textBox2.Text);
            //Выбор функции
            if (listBox1.SelectedIndex == 1) fu = "cos";
            if (listBox1.SelectedIndex == 2) fu = "tn";
            // ТОЧНОСТЬ ВЫЧИСЛЕНИЙ
            if (radioButton1.Checked)
            {
                ff = "F3";
            }
            else
            if (radioButton2.Checked)
            {
                ff = "F4";
            } else
            if (radioButton3.Checked)
            {
                ff = "F5";
            };
            switch (fu)
            {
                case "sin": textBox1.Text = " sin= " +
                    Math.Sin(x).ToString(ff); break;
                case "cos": textBox1.Text = " cos= " +
                    Math.Cos(x).ToString(ff); break;
                case "tn": textBox1.Text = " tn= " +
                    (Math.Sin(x) / Math.Cos(x)).ToString(ff); break;
            }
        }
    }
}

```

Работа программы:

The screenshot shows a Windows application window titled "Form1". Inside the window, there are several controls:

- Вычисляемая функция (Calculating function):** A list box containing "sin", "cos", and "tn". The "cos" option is currently selected.
- Введите значение переменной (Enter variable value):** A text input field containing the number "2".
- Точность вычислений (Calculation accuracy):** Three radio buttons with labels "0.001", "0.0001", and "0.00001". The "0.0001" option is selected.
- Результаты вычислений (Calculation results):** A text area displaying the result "cos= -0,4161".
- Вычисление функции (Calculate function):** A button located at the bottom right of the form.

Рисунок 2.1 – Работа программы вычисления функции

Работа программы очевидна и не нуждается в дополнительных комментариях.

Другие элементы управления – меню, диалоговые окна, рисунки и т.д. будут рассмотрены в следующих лекциях дисциплины.