Лекция. Обзор элементов управления и их свойств.

Windows Forms позволяет разрабатывать приложения с полнофункциональным графическим интерфейсом. Для работы с WF необходимо создать проект. Основные окна проекта:

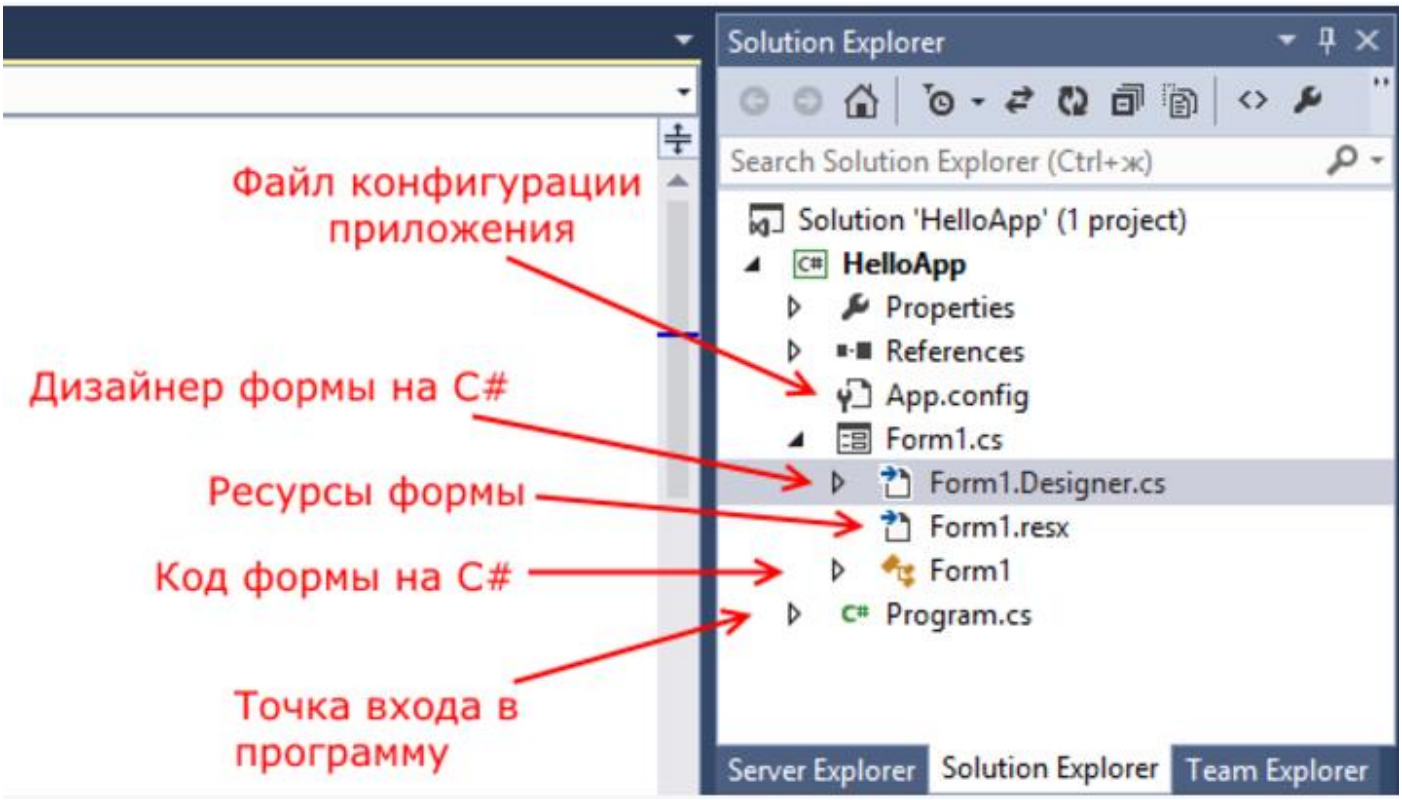
Обозреватель решений – показывает структуру проекта;

Графический дизайнер формы – позволяет создавать интерфейс;

Панель элементов – готовые компоненты для оформления интерфейса;

Окно свойств – позволяет настроить свойства элементов интерфейса.

Рассмотрим структуру проекта. Несмотря на то, что при создании проекта мы видим только главную форму, проект имеет и несколько компонентов:



Стартовой точкой входа в графическое приложение является класс Program, расположенный в файле *Program.cs*

Сначала программой запускается данный класс, затем с помощью выражения Application.Run(new Form1()) он запускает форму Form1. Если вдруг мы захотим изменить стартовую форму в приложении на какую-нибудь другую, то нам надо изменить в этом выражении Form1 на соответствующий класс формы.

В структуре проекта есть файл *Form1.Designer.cs.* Здесь объявляется частичный класс формы Form1, которая имеет два метода: Dispose(), который выполняет роль деструктора объекта, и InitializeComponent(), который устанавливает начальные значения свойств формы.

При добавлении элементов управления, например, кнопок, их описание также добавляется в этот файл.

Но на практике мы редко будем сталкиваться с этим классом, так как они выполняет в основном дизайнерские функции - установка свойств объектов, установка переменных.

Еще один файл - *Form1.resx* - хранит ресурсы формы. Как правило, ресурсы используются для создания однообразных форм сразу для нескольких языковых культур.

И более важный файл - *Form1.cs*, который в структуре проекта называется просто Form1, содержит код или программную логику формы. По умолчанию здесь есть только конструктор формы, в котором просто вызывается метод InitializeComponent(), объявленный в файле дизайнера *Form1.Designer.cs*. Именно с этим файлом мы и будем больше работать.

ФОРМЫ

Основным объектом для построения интерфейса являются формы. Каждой графической форме соответствует некоторый код, который представляет из себя класс с таким же именем как и имя формы, являющийся потомком класса Form. Также необходимо подключение пространства имен System.Windows.Forms

```
using System;
using System.Windows.Forms;

namespace WindowsFormsApplication5
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
    }
}
```

Основные свойства формы:

Большинство этих свойств оказывает влияние на визуальное отображение формы. Пробежимся по основным свойствам:

· **Name**: устанавливает имя формы - точнее имя класса, который наследуется от класса Form

· **BackColor**: указывает на фоновый цвет формы. Щелкнув на это свойство, мы сможем выбрать тот цвет, который нам подходит из списка предложенных цветов или цветовой палитры

· **BackgroundImage**: указывает на фоновое изображение формы

· **BackgroundImageLayout**: определяет, как изображение, заданное в свойстве BackgroundImage, будет располагаться на форме.

· **ControlBox**: указывает, отображается ли меню формы. В данном случае под меню понимается меню самого верхнего уровня, где находятся иконка приложения, заголовок формы, а также кнопки минимизации формы и крестик. Если данное свойство имеет значение false, то мы не увидим ни иконку, ни крестика, с помощью которого обычно закрывается форма

Cursor: определяет тип курсора, который используется на форме

· **Enabled**: если данное свойство имеет значение false, то она не сможет получать ввод от пользователя, то есть мы не сможем нажать на кнопки, ввести текст в текстовые поля и т.д.

· **Font**: задает шрифт для всей формы и всех помещенных на нее элементов управления. Однако, задав у элементов формы свой шрифт, мы можем тем самым переопределить его

· **ForeColor**: цвет шрифта на форме

· **FormBorderStyle**: указывает, как будет отображаться граница формы и строка заголовка. Устанавливая данное свойство в None можно создавать внешний вид приложения произвольной формы

· **HelpButton**: указывает, отображается ли кнопка справки формы

· **Icon**: задает иконку формы

· **Location**: определяет положение по отношению к верхнему левому углу экрана, если для свойства StartPosition установлено значение Manual

· **MaximizeBox**: указывает, будет ли доступна кнопка максимизации окна в заголовке формы

· **MinimizeBox**: указывает, будет ли доступна кнопка минимизации окна

· **MaximumSize**: задает максимальный размер формы

· **MinimumSize**: задает минимальный размер формы

· **Opacity**: задает прозрачность формы

· **Size**: определяет начальный размер формы

· **StartPosition**: указывает на начальную позицию, с которой форма появляется на экране

· **Text**: определяет заголовок формы

· **TopMost**: если данное свойство имеет значение true, то форма всегда будет находиться поверх других окон
· **Visible**: видима ли форма, если мы хотим скрыть форму от пользователя, то можем задать данному свойству значение false
· **WindowState**: указывает, в каком состоянии форма будет находиться при запуске: в нормальном, максимизированном или минимизированном
Мы можем изменять свойства формы непосредственно в окне свойств, а можем изменять их программно. Для этого в программном коде после инициализации свойств можно прописать и собственные значения:

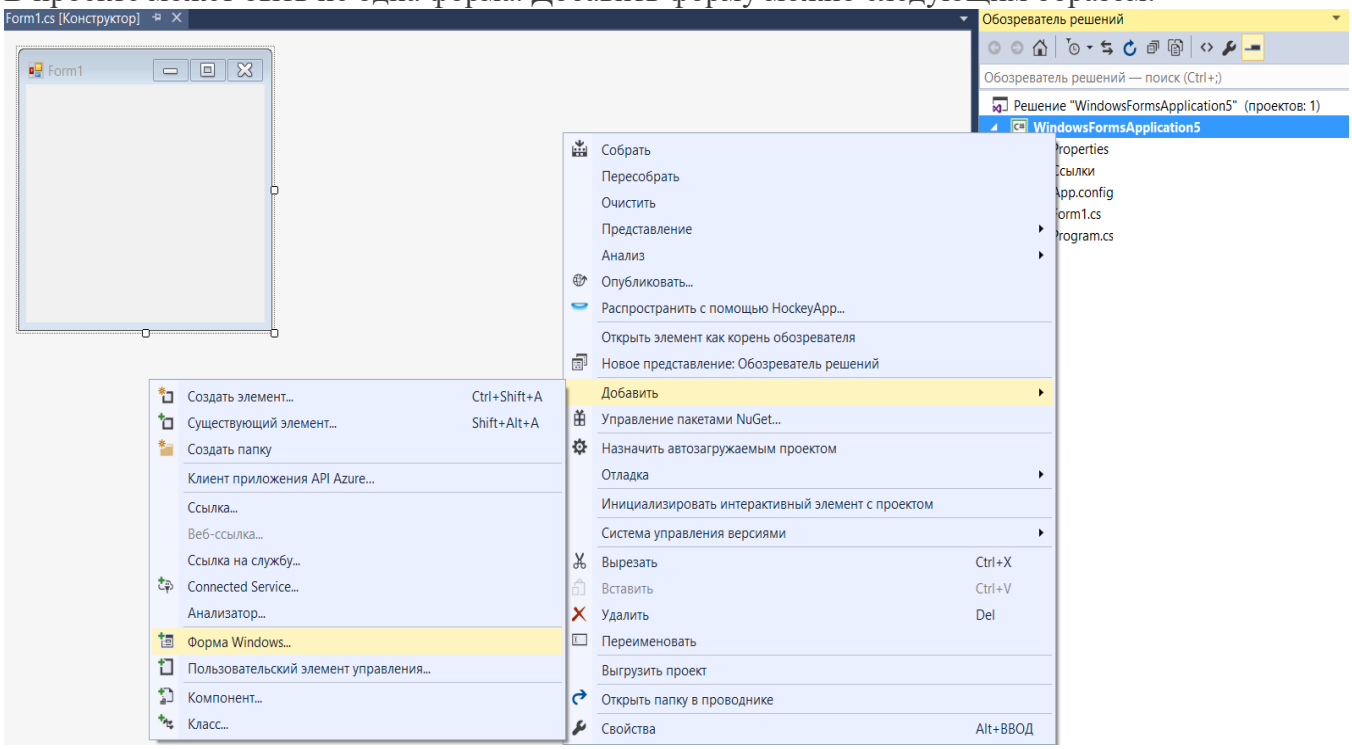
```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace HelloApp
{
        public partial class Form1 : Form
        {
        public Form1()
        {
        InitializeComponent();
        Text = "Hello World!";
        this.BackColor = Color.Aquamarine;
        this.Width = 250;
        this.Height = 250;
        //размер
        this.Size = new Size(200,150);
        //начальная позиция
        this.StartPosition = FormStartPosition.CenterScreen;
        //фон и цвета формы
        this.BackColor = Color.Aquamarine;
        this.ForeColor = Color.Red;
        //изображение в качестве фона
        this.BackgroundImage = Image.FromFile("C:\Users\Eugene\Pictures\3332.jpg");
        }
        }
}
```

Кроме свойств компонент с программированием в WF тесно связан механизм событий. Он служит для связи с пользователем программы.
В WinForms есть некоторый стандартный набор событий, который по большей части имеется у всех визуальных компонентов.
Чтобы посмотреть все события элемента, нам надо выбрать этот элемент в поле графического дизайнера и перейти к вкладке событий на панели форм.
В системе Windows к событиям можно отнести нажатие кнопкой мыши, нажатие клавиши и т.п.
Каждому событию соответствует некоторая функция, код которой выполняется в случае наступления события. Добавление же обработчика, созданного таким образом, производится в файле Form1.Designer.cs. Поэтому если мы захотим удалить созданный подобным образом обработчик, то нам надо не только удалить метод из кода формы в Form1.cs, но и удалить добавление обработчика в этом файле.

Рассмотрим основные события формы.
Load – Происходит до первоначального отображения формы, т.е. при загрузке формы.
FormClosed – Происходит после закрытия формы.
FormClosing – Происходит до закрытия формы.
Click – Происходит при щелчке мыши
DoubleClick – Происходит при двойном щелчке мыши.

Добавление форм.

В проекте может быть не одна форма. Добавить форму можно следующим образом:



<u>Взаимосвязь между формами</u> организовывают следующим образом:

В форме-родителе в соответствующем обработчике события создают объект класса дочерней формы и применяют метод Show()
Form2 newForm = new Form2();
newForm.Show();
Теперь перейдем к коду первой формы, где мы вызывали вторую форму и изменим его на следующий:

```
private void button1_Click(object sender, EventArgs e)
{
Form2 newForm = new Form2(this);
newForm.Show();
}
```

При работе с несколькими формами надо учитывать, что одна из них является главной – которая запускается первой в файле Program.cs. Если у нас одновременно открыта куча форм, то при закрытии главной закрывается все приложение и вместе с ним все остальные формы.

Элементы управления

Элементы управления представляют собой визуальные классы, которые получают введенные пользователем данные и могут инициировать различные события. Все элементы управления наследуются от класса Control и поэтому имеют ряд общих свойств:

- **Anchor**: Определяет, как элемент будет растягиваться
- **BackColor**: Определяет фоновый цвет элемента
- **BackgroundImage**: Определяет фоновое изображение элемента
- **ContextMenu**: Контекстное меню, которое открывается при нажатии на элемент правой кнопкой мыши. Задается с помощью элемента ContextMenu
- **Cursor**: Представляет, как будет отображаться курсор мыши при наведении на элемент
- **Dock**: Задает расположение элемента на форме
- **Enabled**: Определяет, будет ли доступен элемент для использования. Если это свойство имеет значение False, то элемент блокируется.
- **Font**: Устанавливает шрифт текста для элемента
- **ForeColor**: Определяет цвет шрифта
- **Location**: Определяет координаты верхнего левого угла элемента управления
- **Name**: Имя элемента управления
- **Size**: Определяет размер элемента
- **Width**: ширина элемента
- **Height**: высота элемента
- **TabIndex**: Определяет порядок обхода элемента по нажатию на клавишу Tab
- **Tag**: Позволяет сохранять значение, ассоциированное с этим элементом управления

Кнопка

Наиболее часто используемым элементом управления является кнопка. Обрабатывая событие нажатия кнопки, мы может производить те или иные действия.

При нажатии на кнопку на форме в редакторе Visual Studio мы по умолчанию попадаем в код обработчика события Click, который будет выполняться при нажатии:

```
private void button1_Click(object sender, EventArgs e)
{
    MessageBox.Show("Hello World");
}
```

Оформление кнопки

Чтобы управлять внешним отображением кнопки, можно использовать свойство **FlatStyle**. Оно может принимать следующие значения:

- **Flat** - Кнопка имеет плоский вид
- **Popup** - Кнопка приобретает объемный вид при наведении на нее указателя, в иных случаях она имеет плоский вид
- **Standard** - Кнопка имеет объемный вид (используется по умолчанию)
- **System** - Вид кнопки зависит от операционной системы

Изображение на кнопке

Как и для многих элементов управления, для кнопки можно задавать изображение с помощью свойства BackgroundImage. Однако мы можем также управлять размещением текста и изображения на кнопки. Для этого надо использовать свойство **TextImageRelation**. Оно приобретает следующие значения:

- **Overlay**: текст накладывается на изображение
- **ImageAboveText**: изображение располагается над текстом

- **TextAboveImage**: текст располагается над изображением
- **ImageBeforeText**: изображение располагается перед текстом
- **TextBeforeImage**: текст располагается перед изображением

Текстовое поле TextBox

Для ввода и редактирования текста предназначены текстовые поля - элемент TextBox. Так же как и у элемента Label текст элемента TextBox можно установить или получить с помощью свойства Text.

По умолчанию при переносе элемента с панели инструментов создается однострочное текстовое поле. Для отображения больших объемов информации в текстовом поле нужно использовать его свойства `Multiline` и `ScrollBars`. При установке для свойства `Multiline` значения true, все избыточные символы, которые выходят за границы поля, будут переноситься на новую строку.

Кроме того, можно сделать прокрутку текстового поля, установив для его свойства `ScrollBars` одно из значений:

- **None**: без прокруток (по умолчанию)
- **Horizontal**: создает горизонтальную прокрутку при длине строки, превышающей ширину текстового поля
- **Vertical**: создает вертикальную прокрутку, если строки не помещаются в текстовом поле
- **Both**: создает вертикальную и горизонтальную прокрутку

Автозаполнение текстового поля

Элемент TextBox обладает достаточными возможностями для создания автозаполняемого поля. Для этого нам надо привязать свойство **AutoCompleteCustomSource** элемента TextBox к некоторой коллекции, из которой берутся данные для заполнения поля.

Итак, добавим на форму текстовое поле и пропишем в код события загрузки следующие строки:

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
        AutoCompleteStringCollection source = new AutoCompleteStringCollection()
        {
            "Кузнецов",
            "Иванов",
            "Петров",
            "Кустов"
        };
        textBox1.AutoCompleteCustomSource = source;
        textBox1.AutoCompleteMode = AutoCompleteMode.SuggestAppend;
        textBox1.AutoCompleteSource = AutoCompleteSource.CustomSource;
    }
}
```

Режим автодополнения, представленный свойством **AutoCompleteMode**, имеет несколько возможных значений:

- **None**: отсутствие автодополнения
- **Suggest**: предлагает варианты для ввода, но не дополняет
- **Append**: дополняет введенное значение до строки из списка, но не предлагает варианты для выбора
- **SuggestAppend**: одновременно и предлагает варианты для автодополнения, и дополняет введенное пользователем значение

Перенос по словам

Чтобы текст в элементе TextBox переносился по словам, надо установить свойство **WordWrap** равным `true`. То есть если одно слово не умещается на строке, то но переносится на следующую. Данное свойство будет работать только для многострочных текстовых полей.

Ввод пароля

Также данный элемент имеет свойства, которые позволяют сделать из него поле для ввода пароля. Так, для этого надо использовать **PasswordChar** и **UseSystemPasswordChar**.

Свойство PasswordChar по умолчанию не имеет значение, если мы установим в качестве него какой-нибудь символ, то этот символ будут отображаться при вводе любых символов в текстовое поле.

Свойство UseSystemPasswordChar имеет похожее действие. Если мы установим его значение в `true`, то вместо введенных символов в текстовом поле будет отображаться знак пароля, принятый в системе, например, точка.

Событие TextChanged

Из всех событий элемента TextBox следует отметить событие `TextChanged`, которое срабатывает при изменении текста в элементе. Например, поместим на форму кроме текстового поля метку и сделаем так, чтобы при изменении текста в текстовом поле также менялся текст на метке:

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

        textBox1.TextChanged += textBox1_TextChanged;
    }

    private void textBox1_TextChanged(object sender, EventArgs e)
    {
        label1.Text = textBox1.Text;
    }
}
```

Элемент PictureBox

PictureBox предназначен для показа изображений. Он позволяет отобразить файлы в формате bmp, jpg, gif, а также метафайлы ищображений и иконки. Для установки изображения в PictureBox можно использовать ряд свойств:

- **Image**: устанавливает объект типа Image
- **ImageLocation**: устанавливает путь к изображению на диске или в интернете
- **InitialImage**: некоторое начальное изображение, которое будет отображаться во время загрузки главного изображения, которое хранится в свойстве Image
- **ErrorImage**: изображение, которое отображается, если основное изображение не удалось загрузить в PictureBox

Чтобы установить изображение в Visual Studio, надо в панели Свойств PictureBox выбрать свойство Image. В этом случае нам откроется окно импорта изображения в проект, где мы собственно и сможем выбрать нужное изображение на компьютере и установить его для PictureBox:
И затем мы сможем увидеть данное изображение в PictureBox:
Либо можно загрузить изображение в коде:

```
pictureBox1.Image = Image.FromFile("C:\Users\Eugene\Pictures\12.jpg");
```

Размер изображения

Для установки изображения в PictureBox используется свойство **SizeMode**, которое принимает следующие значения:

- Normal: изображение позиционируется в левом верхнем углу PictureBox, и размер изображения не изменяется. Если PictureBox больше размеров изображения, то по справа и снизу появляются пустоты, если меньше - то изображение обрезается
- StretchImage: изображение растягивается или сжимается таким обраом, чобы вместиться по всей ширине и высоте элемента PictureBox
- AutoSize: элемент PictureBox автоматически растягивается, подстраиваясь под размеры изображения
- CenterImage: если PictureBox меньше изображения, то изображение обрезается по краям и выводится только его центральная часть. Если же PictureBox больше изображения, то оно позиционируется по центру.
- Zoom: изоражение подстраивается под размеры PictureBox, сохраняя при этом пропорции

Элемент ComboBox

Элемент ComboBox образует выпадающий список и совмещает функциональность компонентов ListBox и TextBox. Для хранения элементов списка в ComboBox также предназначено свойство **Items**. Подобным образом, как и с ListBox, мы можем в окне свойств на свойство Items и нам отобразится окно для добавления элементов ComboBox:
И как и с компонентом ListBox, здесь мы также можем программно управлять элементами.
Добавление элементов:

```
// добавляем один элемент
comboBox1.Items.Add("Парагвай");
// добавляем набор элементов
comboBox1.Items.AddRange(new string[] { "Уругвай", "Эквадор" });
// добавляем один элемент на определенную позицию
comboBox1.Items.Insert(1, "Боливия");
```

При добавлении с помощью методов Add / AddRange все новые элементы помещаются в конец списка. Однако если мы зададим у ComboBox свойство Sorted равным true, тогда при добавлении будет автоматически производиться сортировка.
Удаление элементов:

```
// удаляем один элемент
comboBox1.Items.Remove("Аргентина");
// удаляем элемент по индексу
comboBox1.Items.RemoveAt(1);
// удаляем все элементы
comboBox1.Items.Clear();
```

Мы можем получить элемент по индексу и производить с ним разные действия. Например, изменить его:

```
comboBox1.Items[0] = "Парагвай";
```

Настройка оформления ComboBox

С помощью ряда свойств можно настроить стиль оформления компонента. Так, свойство **DropDownWidth** задает ширину выпадающего списка. С помощью свойства **DropDownHeight** можно установить высоту выпадающего списка.
Еще одно свойство **MaxDropDownItems** позволяет задать число видимых элементов списка - от 1 до 100. По умолчанию это число равно 8.

Другое свойство **DropDownStyle** задает стиль ComboBox. Оно может принимать три возможных значения:

- **Dropdown**: используется по умолчанию. Мы можем открыть выпадающий список вариантов при вводе значения в текстовое поле или нажав на кнопку со стрелкой в правой части элемента, и нам отобразится собственно выпадающий список, в котором можно выбрать возможный вариант
- **DropdownList**: чтобы открыть выпадающий список, надо нажать на кнопку со стрелкой в правой стороне элемента
- **Simple**: ComboBox представляет простое текстовое поле, в котором для перехода между элементами мы можем использовать клавиши клавиатуры вверх/вниз

Событие SelectedIndexChanged

Наиболее важным событием для ComboBox также является событие SelectedIndexChanged, позволяющее отследить выбор элемента в списке:

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
        comboBox1.SelectedIndexChanged += comboBox1_SelectedIndexChanged;
    }
    void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
    {
        string selectedState = comboBox1.SelectedItem.ToString();
        MessageBox.Show(selectedState);
    }
}
```

Здесь также свойство SelectedItem будет ссылаться на выбранный элемент.