**Department of Economic Informatics and Cybernetics**
Bucharest University of Economic Studies

# Windows Applications Programming

## Windows Forms

Liviu-Adrian Cotfas, PhD.

liviu.cotfas@ase.ro

# Few words about me…

https://ro.linkedin.com/in/cotfasliviu

# Further Reading / Watching

- Courses on **Microsoft Virtual Academy** - mva.microsoft.com
  - Free

- Courses on **PluralSight** - www.pluralsight.com
  - Free trial
  - Free access (limited period) through Microsoft DreamSpark

# API reference and Source code

- API reference:
  - https://msdn.microsoft.com/en-us/library/

- .NET Framework source code:
  - http://referencesource.microsoft.com/#mscorlib/system/string.cs,8281103e6f23cb5c

# Windows Forms

# .NET Graphical User Interface

- Windows Forms (2001)
- Windows Presentation Foundation
- Universal Windows Platform

# Startup and Shutdown

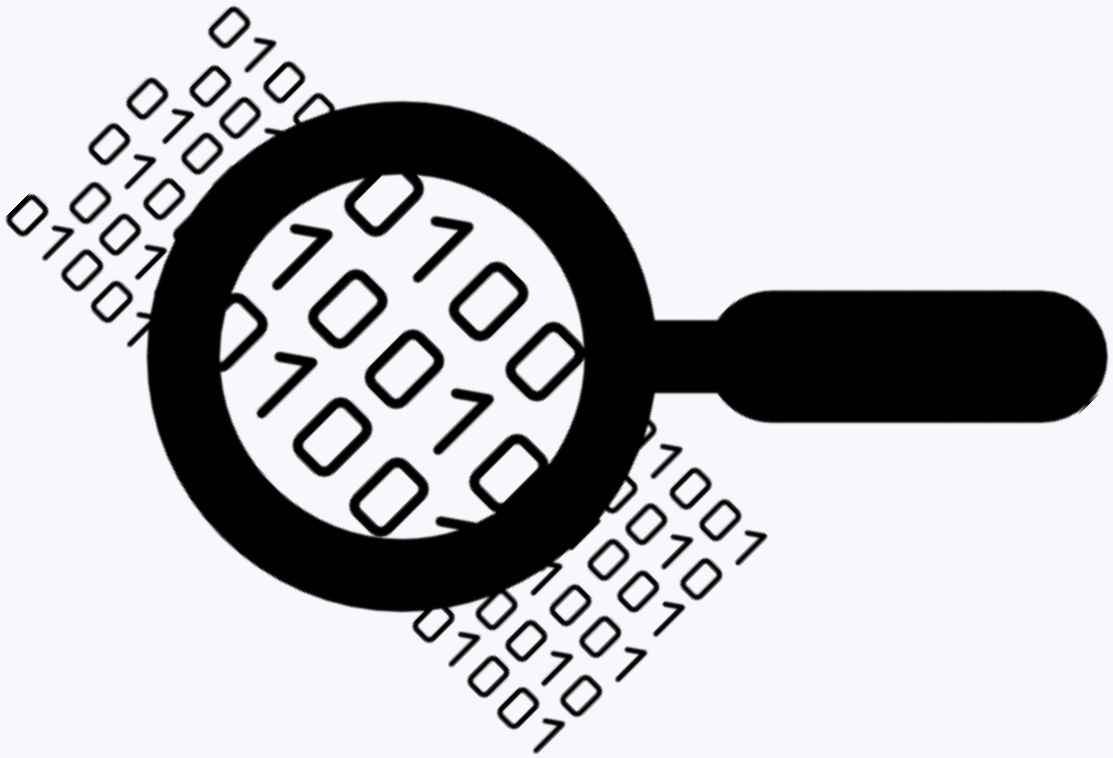
- Startup

```
[STAThread]
static void Main( )
{
        Application.Run(new Form1( ));
}
```

# Application Class

- The Application Class
  - Run
  - Exit
  - ThreadException

# Demo



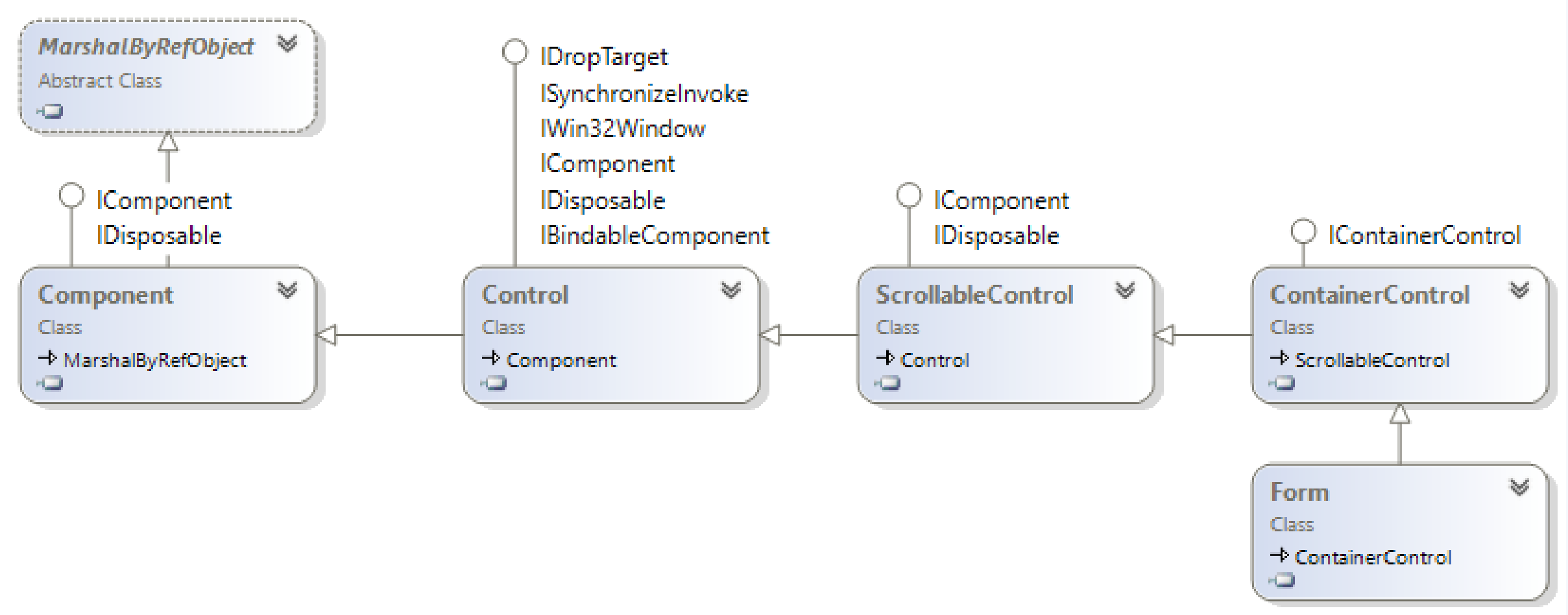
# Form Class

# Form Class

- All windows in a Windows Forms application are represented by objects of some type deriving from the Form class.

```
public partial class MainForm : Form
{
        // Constructor
        public MainForm()
        {
                InitializeComponent();
        }
}
```

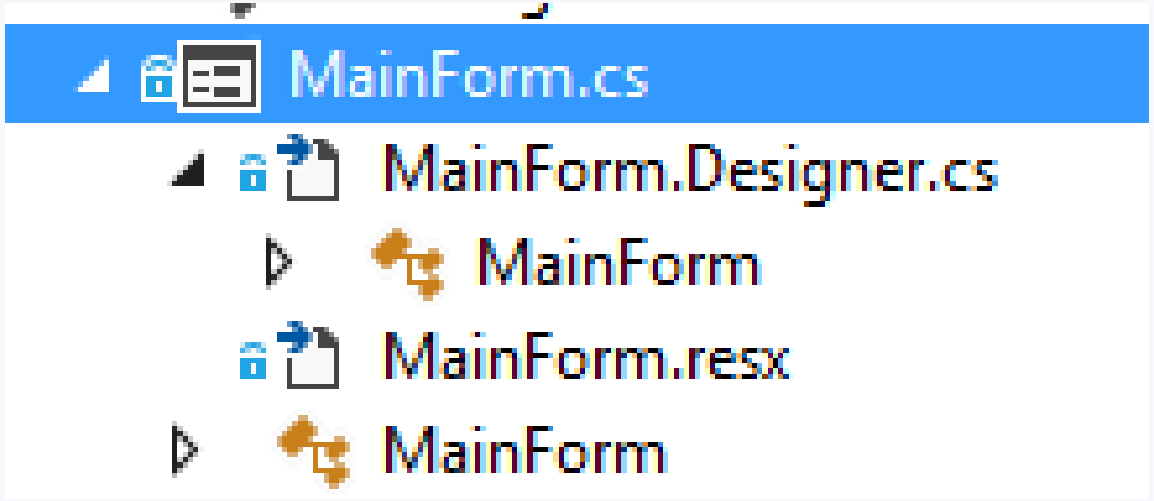
# Inheritance

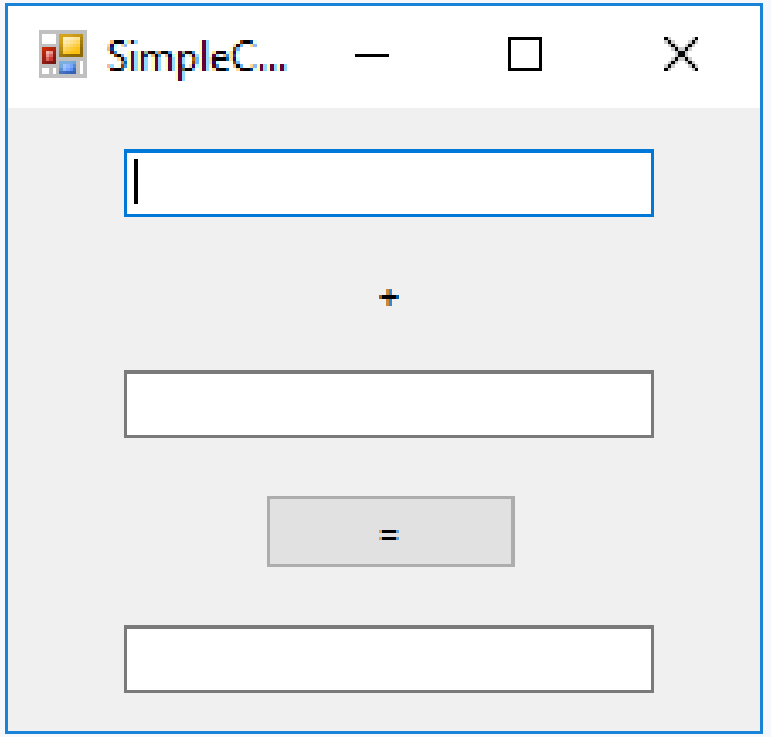



Further reading: link

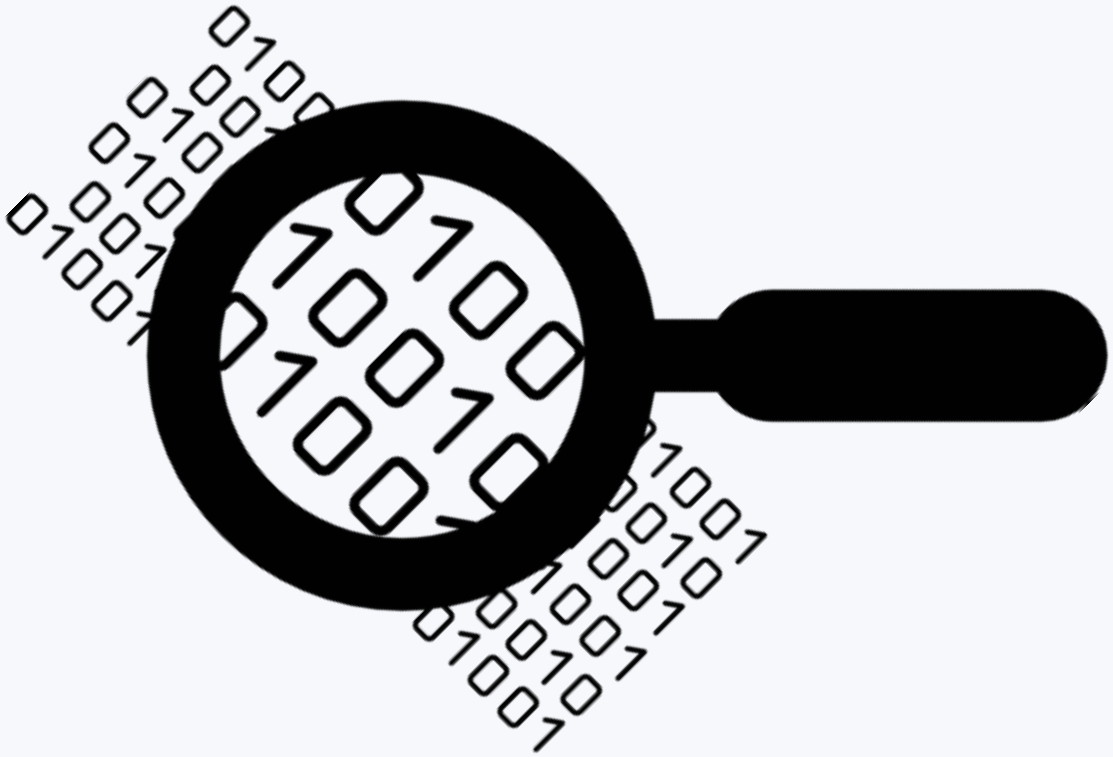
# The Forms Designer

- visual designer that auto-generates code.
- Uses **partial classes**.



# Demo



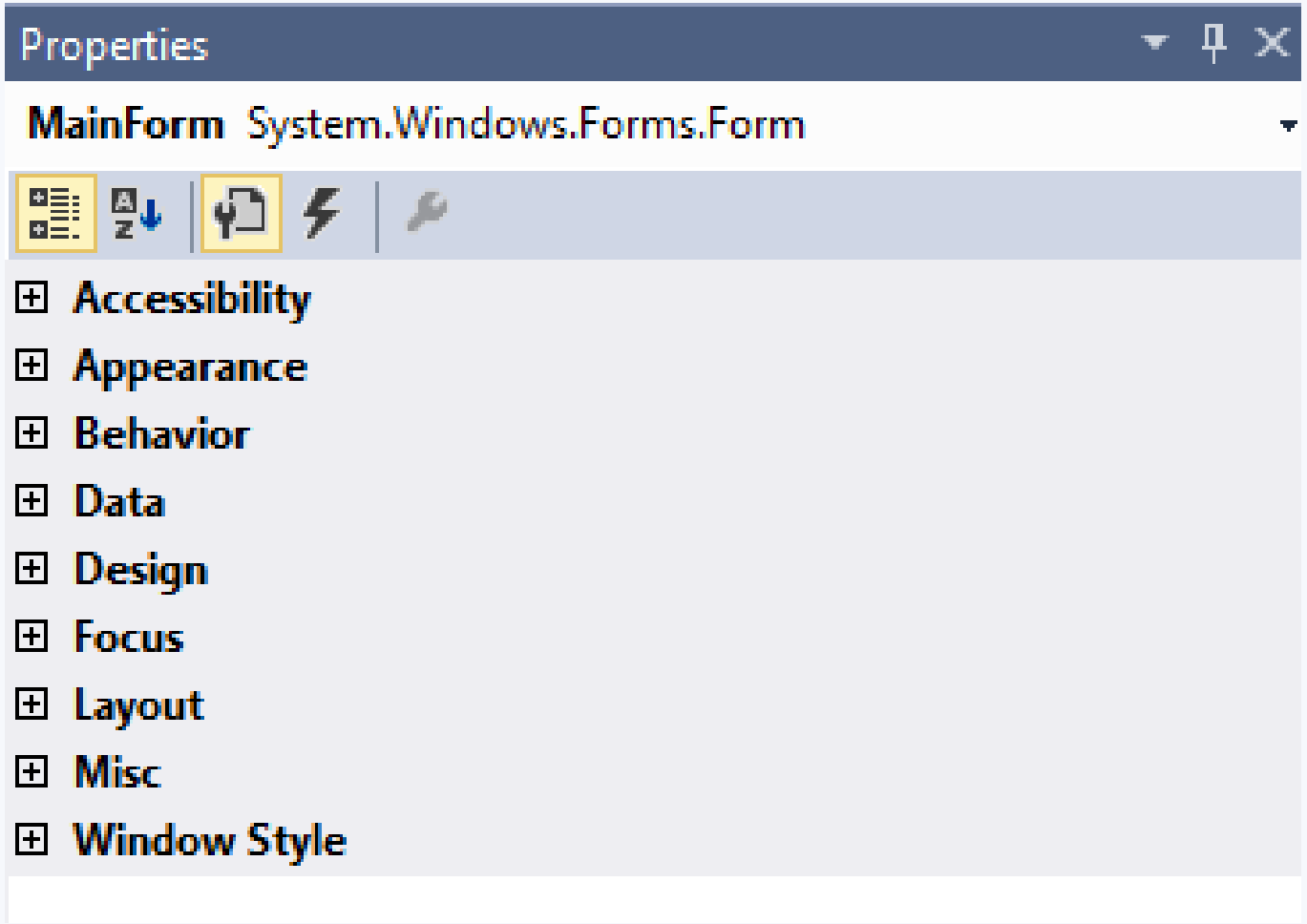
# Partial Class

- It is possible to split the definition of a class or a struct, an interface or a method over two or more source files. Each source file contains a section of the type or method definition, and all parts are combined when the application is compiled.

```
public partial class Employee{
    public void DoWork()    {
    }
}

public partial class Employee{
    public void GoToLunch()    {
    }
}
```

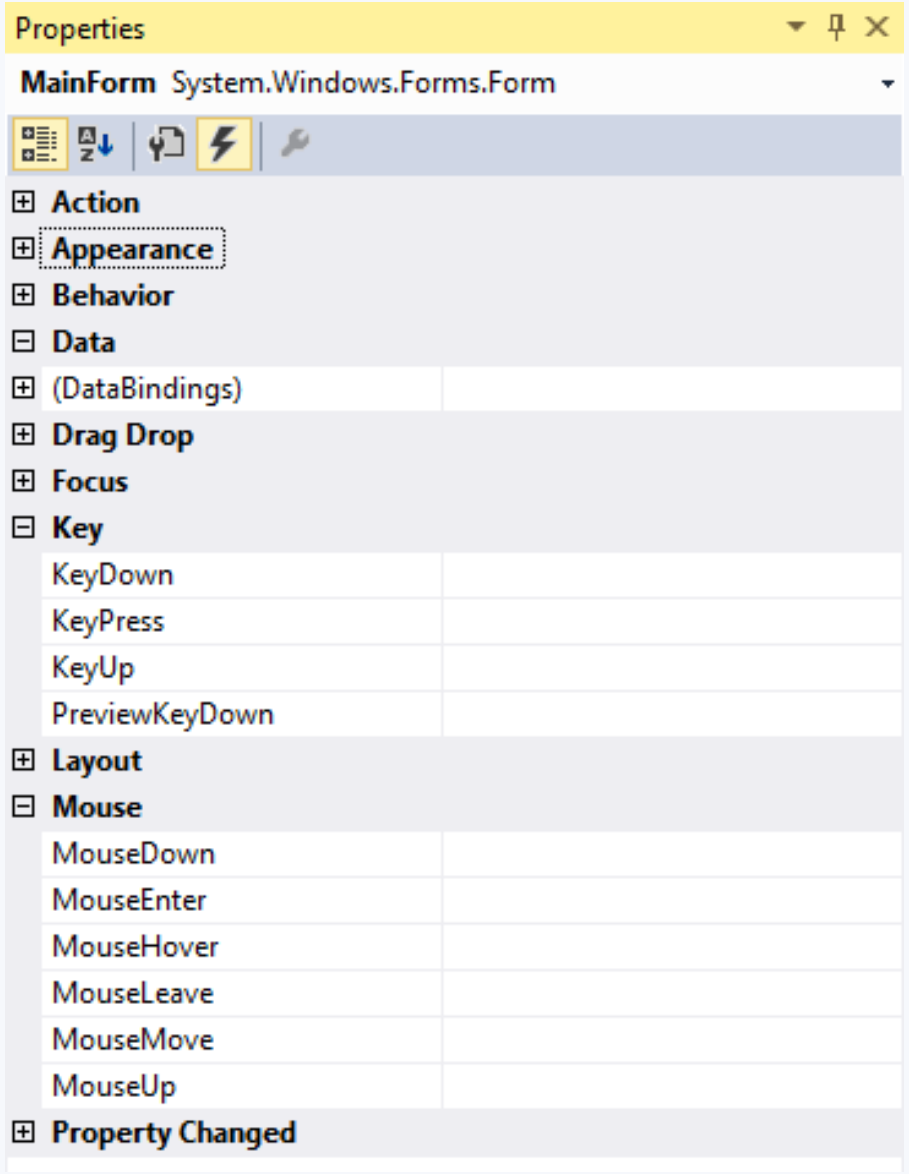
# Properties

- Appearance
  - BackColor
  - Font
  - ForeColor
  - Text
- Design
  - Name



# Events

- Key
  - KeyDown
  - KeyPress
  - KeyUp
- Mouse
  - MouseDown
  - MouseUp
  - Click



# Keyboard Events

# Keyboard Events

- Key events occur in the order specified bellow.

| Event | Event data | Description |
|---|---|---|
| KeyDown | KeyEventArgs | Raised when a key is pressed. The KeyDown event occurs prior to the KeyPress event. |
| KeyPress | KeyPressEventArgs | Raised when a character generating key is pressed. The KeyPress event occurs after the KeyDown event and before the KeyUp event. |
| KeyUp | KeyEventArgs | Raised when a key is released. |

# KeyPress

- Occurs when a character is pressed on the keyboard, and again each time the character is repeated while it continues to be pressed.

- The KeyPress event is not raised by non-character keys other than space and backspace; however, the non-character keys do raise the KeyDown and KeyUp events.

- Use the KeyChar property to sample keystrokes at run time and to consume or modify a subset of common keystrokes.

# KeyPressEventArgs

| Property | Description |
| --- | --- |
| Handled | Boolean value indicating if the event was handled. false until set otherwise. When true, the keystroke is not displayed. |
| KeyChar | Read-only value of type char containing the composed ASCII character. |

# KeyDown and KeyUp

- **KeyDown** - Occurs when a key on the keyboard is pressed down.
- **KeyUp** - Occurs when a key on the keyboard is released.
- The **KeyDown** and **KeyUp** events are useful to fine-tune an application's behavior as keyboard keys are pressed and released, and for handling noncharacter keys such as the function or arrow keys.
- Handlers for these events receive an instance of the KeyEventArgs class as their event parameter.

# KeyEventArgs properties

| Property | Data type | Description |
|---|---|---|
| Alt | Boolean | Read-only value indicating if the Alt key was pressed. true if pressed, false otherwise. |
| Control | Boolean | Read-only value indicating if the Ctrl key was pressed. true if pressed, false otherwise. |
| Shift | Boolean | Read-only value indicating if the Shift key was pressed. true if pressed, false otherwise. |
| Modifiers | Keys | Read-only flags indicating the combination of modifier keys (Alt, Ctrl, Shift) pressed. Modifier keys can be combined using the bitwise OR operator. |
| Handled | Boolean | Value indicating if the event was handled. false until set otherwise. |
| KeyCode | Keys | Read-only value containing the key code for the key pressed. Typical values include the A key, Alt, and BACK (backspace). |
| KeyData | Keys | Read-only value containing the key code for the key pressed, combined with modifier flags to indicate combination of modifier keys (Alt, Ctrl, Shift). |
| KeyValue | integer | Key code property expressed as a read-only integer. |

# Mouse Events

# Mouse Events

| Event | Event argument | Description |
|---|---|---|
| Click | EventArgs | Raised when the control is clicked. |
| DoubleClick | EventArgs | Raised when the control is double-clicked. |
| MouseEnter | EventArgs | Raised when the mouse cursor enters the control. |
| MouseHover | EventArgs | Raised when the mouse cursor hovers over the control. |
| MouseLeave | EventArgs | Raised when the mouse cursor leaves the control. |
| MouseDown | MouseEventArgs | Raised when the mouse cursor is over the control and a mouse button is pressed. |
| MouseMove | MouseEventArgs | Raised when the mouse cursor is moved over the control. |
| MouseWheel | MouseEventArgs | Raised when the control has focus and the mouse wheel is rotated. |
| MouseUp | MouseEventArgs | Raised when the mouse cursor is over the control and a mouse button is released. |

# Click and MouseClick

Depressing a mouse button when the cursor is over a control typically raises the following series of events from the control:

- MouseDown event.
- Click event.
- MouseClick event.
- MouseUp event.

Click events are logically higher-level events of a control. They are often raised by other actions, such as pressing the ENTER key when the control has focus.

MouseClick contains specific mouse information (button, number of clicks, wheel rotation, or location),

# MouseEventArgs properties

| Property | Description |
|---|---|
| Button | Returns the pressed mouse button. Must be one of the members of the MouseButtons enumeration |
| Clicks | Returns the integer number of times the mouse button was pressed and released. Resets after two clicks. |
| Delta | Returns the signed integer number of detents the mouse wheel was rotated. A positive value indicates that the wheel was rotated forward, i.e., away from the user, and a negative value indicates the wheel was rotated backward, i.e., toward the user. |
| X | The X coordinate, in pixels, of the mouse cursor's hot spot when the button was clicked, relative to the top-left corner of the control. |
| Y | The Y coordinate, in pixels, of the mouse cursor's hot spot when the button was clicked, relative to the top-left corner of the control. |

# Click

```
this.btnCalculate.Click += new System.EventHandler(this.btnCalculate_Click);
```

```
private void btnCalculate_Click(object sender, EventArgs e)
{
        var value1 = int.Parse(tbValue1.Text);
        var value2 = int.Parse(tbValue2.Text);

        var sum = value1 + value2;
        tbSum.Text = sum.ToString();
}
```

# Alt Shortcuts

# Alt Shortcuts

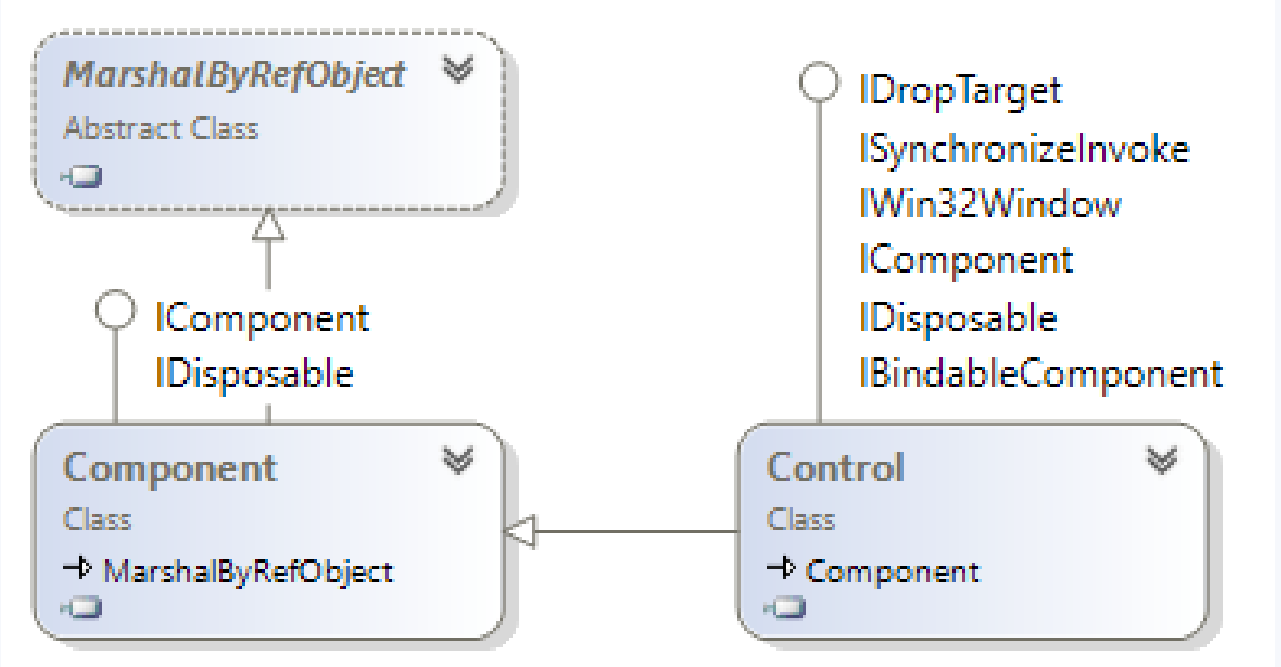
- An **access key** is an underlined character in the text of a menu, menu item, or the label of a control such as a button. It allows the user to "click" a button by pressing the ALT key in combination with the predefined access key.

```
// C#
// Set the letter "P" as an access key.
button1.Text = "&Print";
```

# Controls

# Control Class

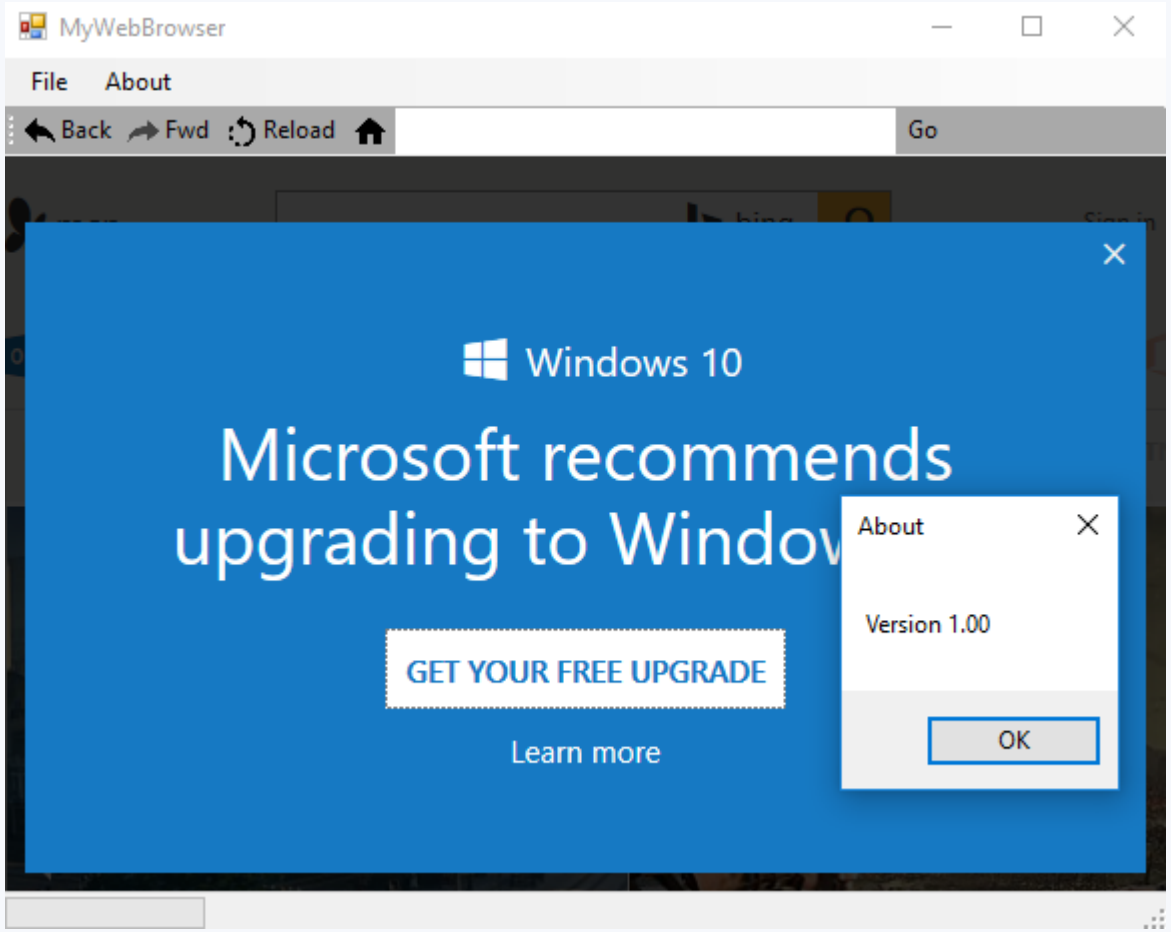
- The Control class forms the basis for all windows controls in .NET, and provides many properties, methods, and events.




- Further reading: link

# Common controls

- Recommended Controls and Components by Function: https://msdn.microsoft.com/en-us/library/xfak08ea%28v=vs.110%29.aspx

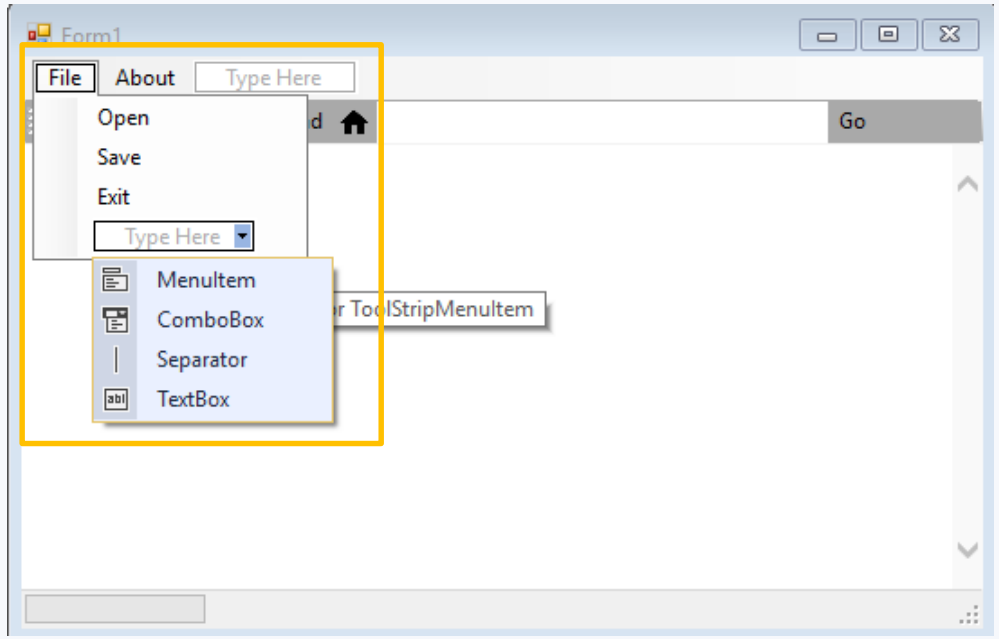


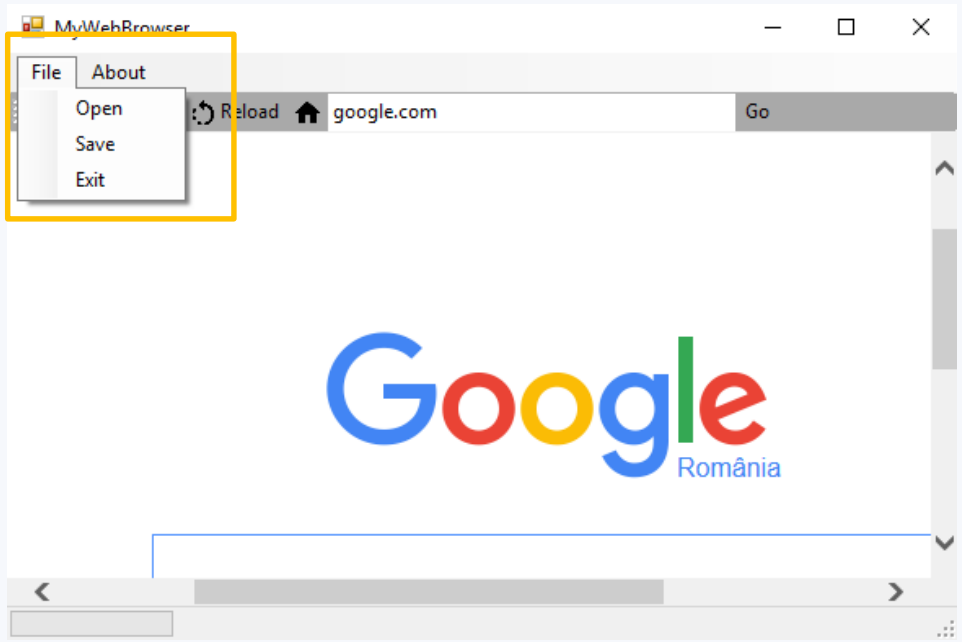
# Common controls

- Alphabetic list of controls and components: https://msdn.microsoft.com/en-us/library/3xdhey7w%28v=vs.110%29.aspx

# Menus

# MenuStrip


Form1
File
About
Type Here
Open
Save
Exit
Type Here
MenuItem
ComboBox
Separator
TextBox
Go


MyWebBrowser
File
About
Open
Save
Exit
Reload
google.com
Go
Google
România

# MenuStrip

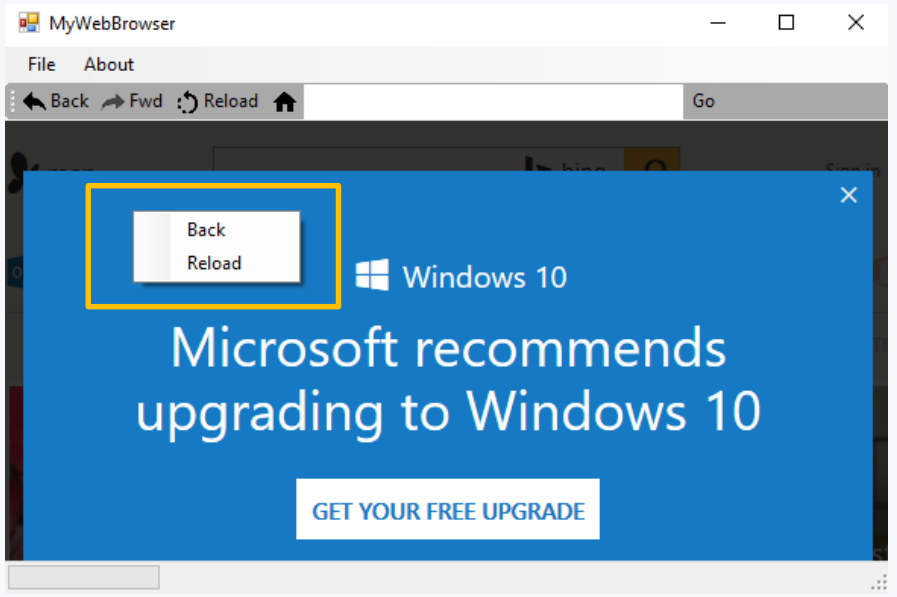
- System.Windows.Forms.**MenuStrip**

- The MenuStrip control represents the container for the menu structure of a form. You can add ToolStripMenuItem objects to the MenuStrip that represent the individual menu commands in the menu structure.

- Each ToolStripMenuItem can be a command for your application or a parent menu for other submenu items.

- Further reading: link

# ContextMenuStrip

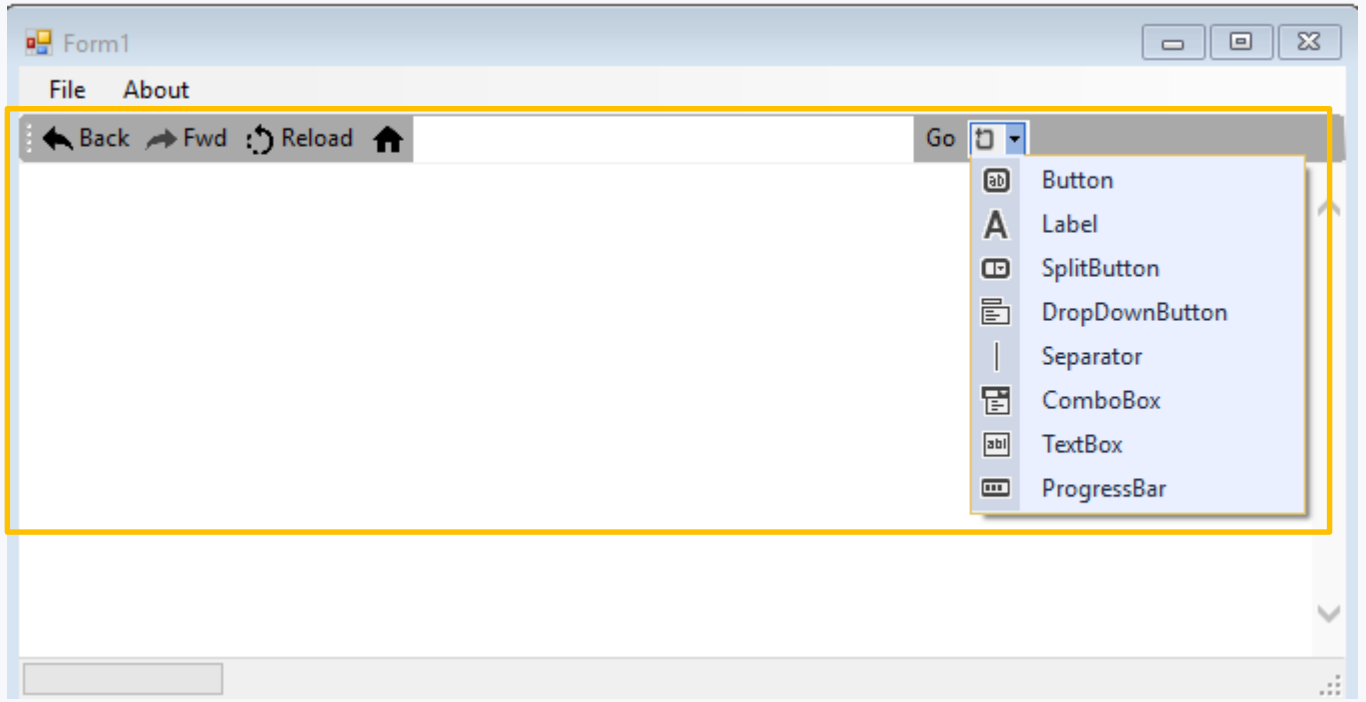
# ContextMenuStrip

- System.Windows.Forms.ContextMenuStrip

- Shortcut menus, also called context menus, appear at the mouse position when the user clicks the right mouse button. Shortcut *menus* provide options for the client area or the control at the mouse pointer location.

- Further reading: link

# ToolStrip


Form1
File
About
Back
Fwd
Reload
Go
Button
Label
SplitButton
DropDownButton
Separator
ComboBox
TextBox
ProgressBar

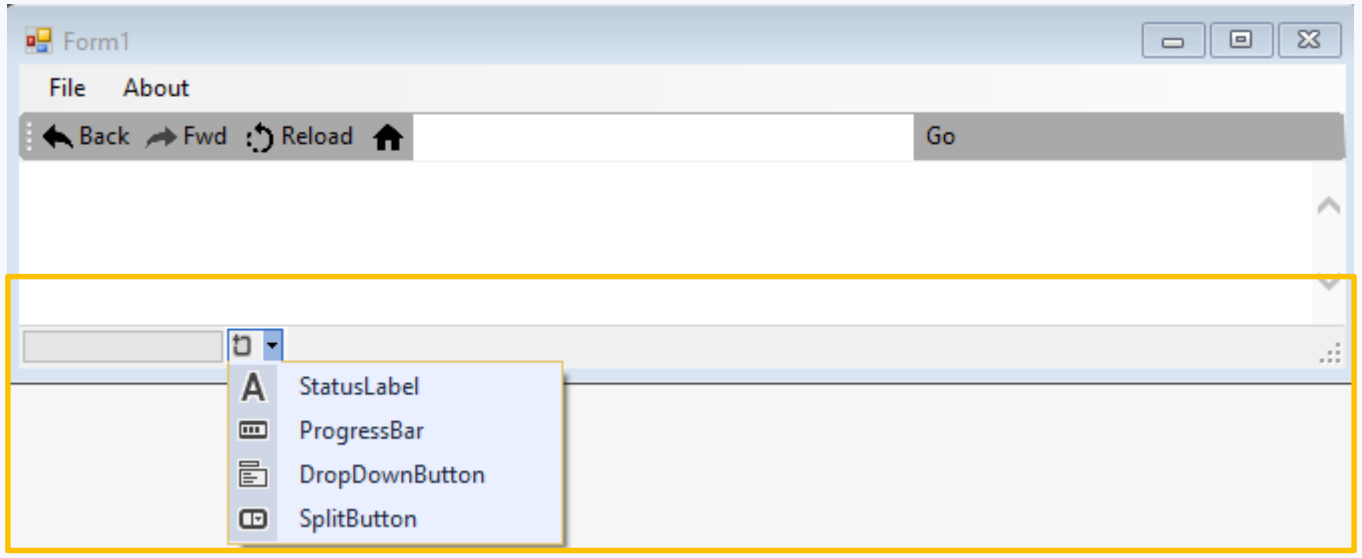
# ToolStrip

- System.Windows.Forms.**ToolStrip**

- The Windows Forms ToolStrip control and its associated classes provide a common framework for combining user interface elements into toolbars, status bars, and menus. ToolStrip controls offer a rich design-time experience that includes in-place activation and editing, custom layout, and rafting, which is the ability of toolbars to share horizontal or vertical space.

- Further reading: link

# StatusStrip



# StatusStrip

- Windows.Forms.**StatusStrip**

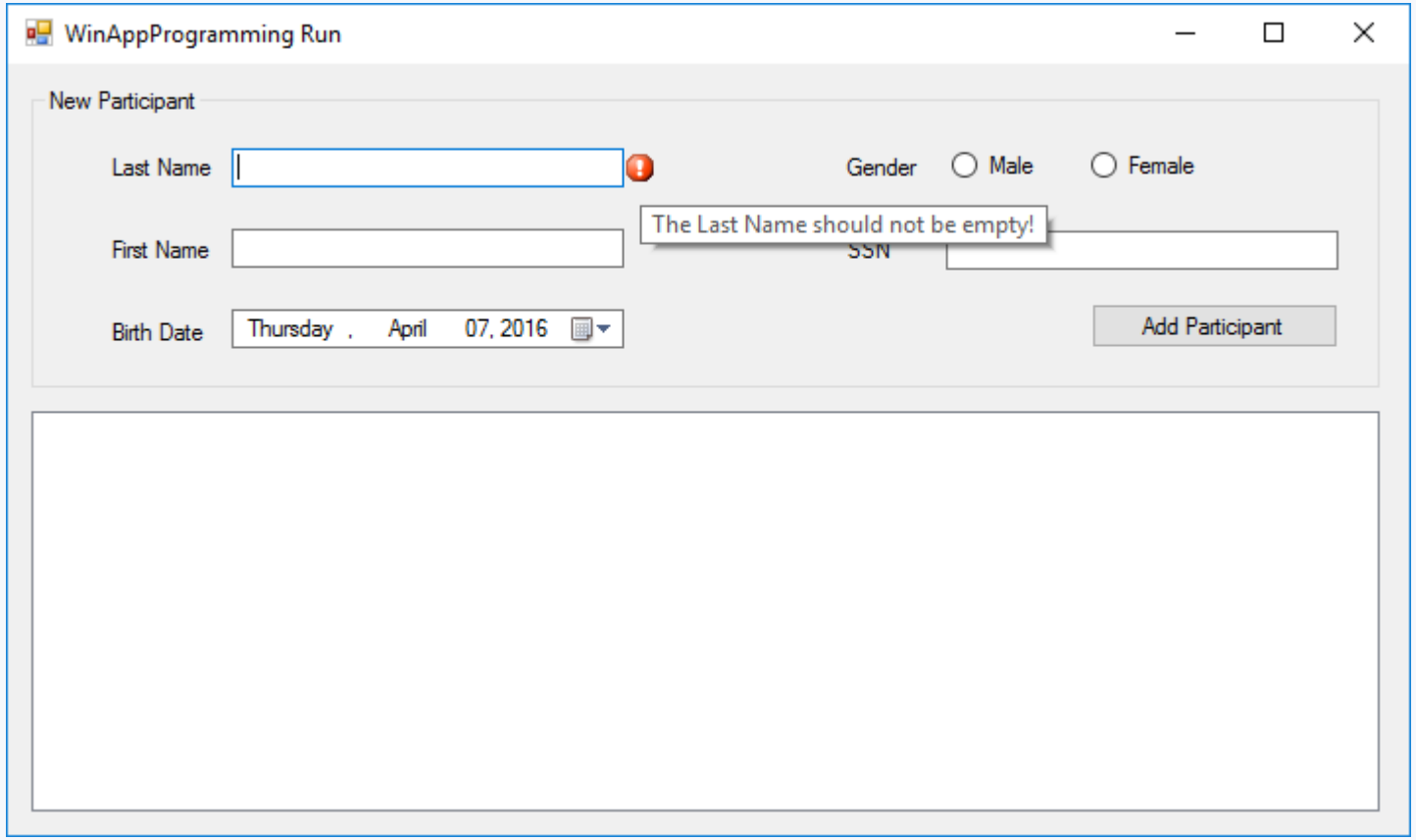
- A StatusStrip control displays information about an object being viewed on a Form, the object's components, or contextual information that relates to that object's operation within your application.

- Typically, a StatusStrip control consists of ToolStripStatusLabel objects, each of which displays text, an icon, or both. The StatusStrip can also contain ToolStripDropDownButton, ToolStripSplitButton, and ToolStripProgressBar controls.

- Further reading: link

# Data Validation

# Using **Validating** and **Validated** Events



# Control.**Validating** Event

- Occurs when the control is validating.

- If the CausesValidation property is set to **false**, the Validating and Validated events are suppressed.

```
private void tbLastName_Validating(object sender, CancelEventArgs e)
{
        string lastName = ((TextBox) sender).Text.Trim();

        if (string.IsNullOrWhiteSpace(lastName))
        {
                e.Cancel = true;

                epLastName.SetError((Control)sender, "Last Name isempty!");
        }
}
```

# Control.**Validating** Event

- Events:
  1. Enter
  2. GotFocus
  3. Leave
  4. Validating
  5. Validated
  6. LostFocus

- If the Cancel property of the CancelEventArgs is set to **true** in the Validating event delegate, all events that would usually occur after the Validating event are suppressed.

# Control.**Validated** Event

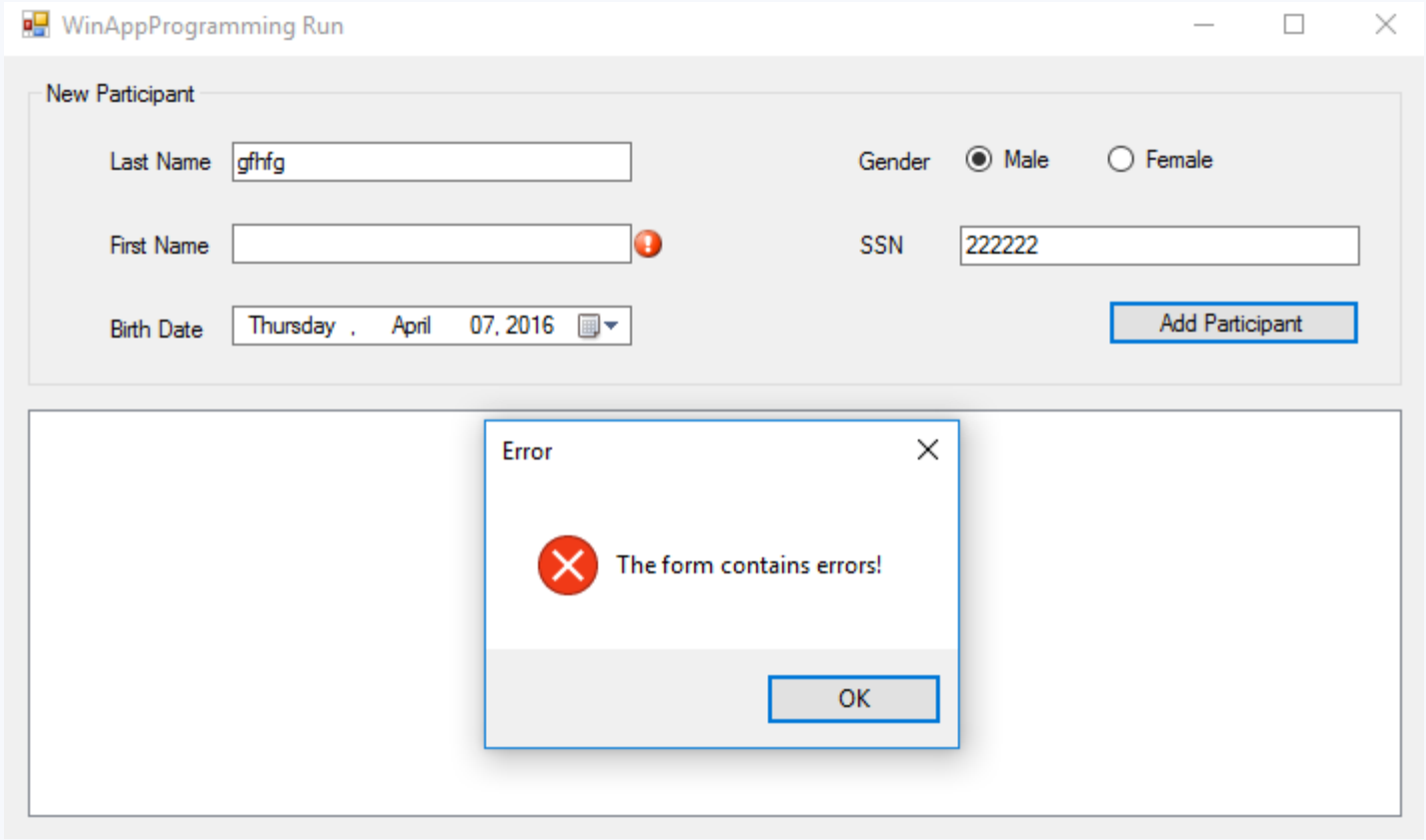
- Occurs when the control is finished validating.

```
private void tbLastName_Validated(object sender, EventArgs e)
{
        epLastName.Clear();
}
```

# Validating the entire form

```
private void btnAdd_Click(object sender, EventArgs e){
        string firstName = tbFirstName.Text.Trim();
        string lastName = tbLastName.Text.Trim();

        bool formContainsErrors = false;

        if (string.IsNullOrWhiteSpace(lastName)){
                epLastName.SetError(tbFirstName, "Last Name is empty!");
                formContainsErrors = true;
        }

        if (string.IsNullOrWhiteSpace(firstName)){…     }

        if (formContainsErrors)     {
                MessageBox.Show("The form contains errors!", "Error",
                        MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
}
```

# Validating the entire form

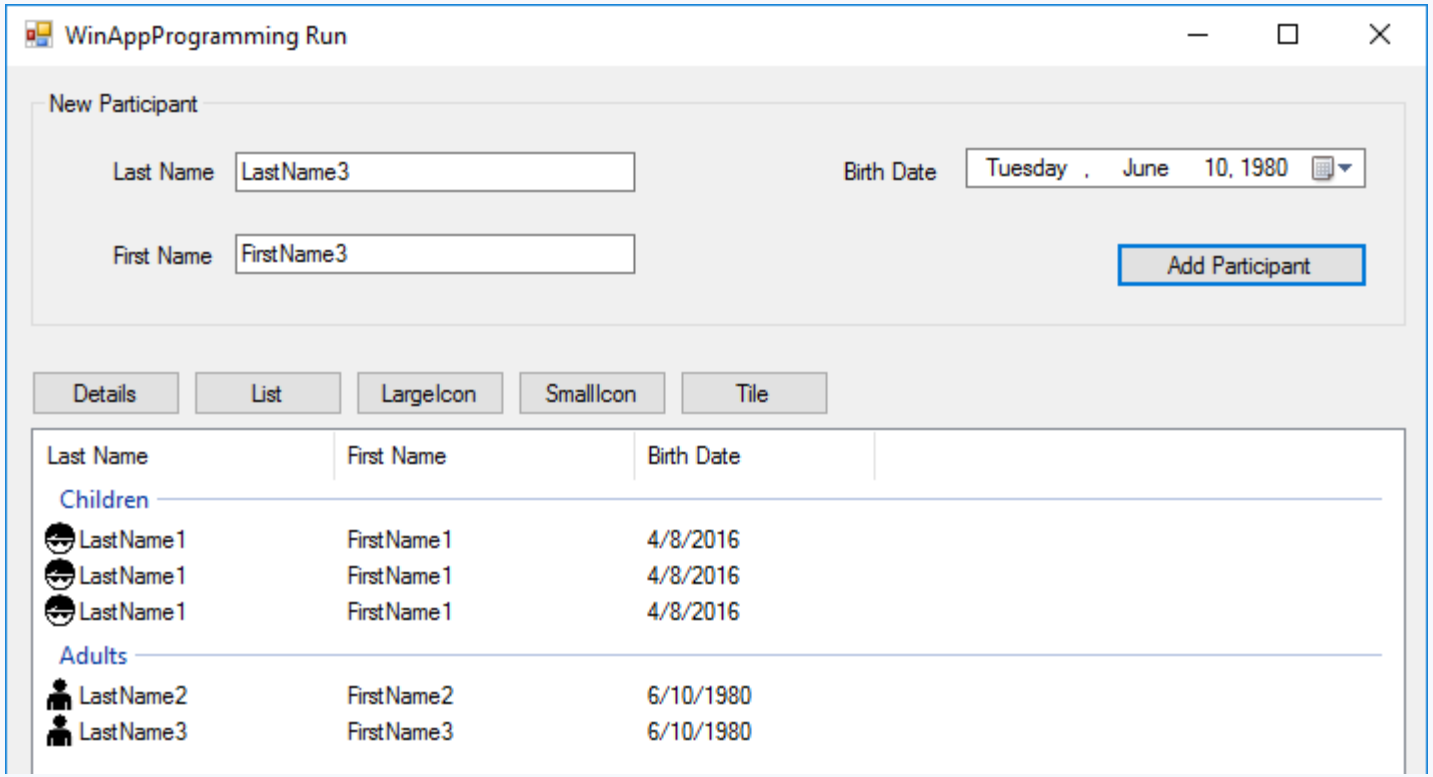


# Complex Visualization Controls

# ListView

- System.Windows.Forms. ListView

- Represents a Windows list view control, which displays a collection of items that can be displayed using one of four different views.



# ListView

- Important properties:
  - Items
  - Columns
  - Groups
  - CheckBoxes
  - **View:** Details, List, Tile, SmallIcon, LargeIcon
  - **FullRowSelect:** True/False
  - **GridLines:** True/False
  - **SmallImageList** / **LargeImageList**

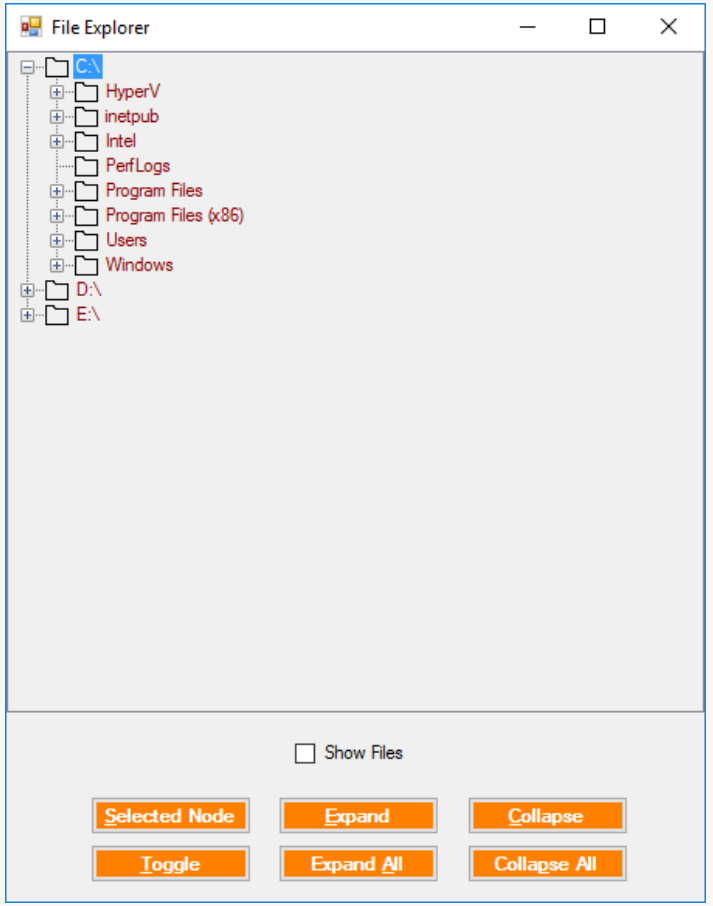
# ListView

- Important events:
  - SelectedIndexChanged

# TreeView

- System.Windows.Forms.**TreeView**

- Displays a hierarchical collection of labeled items, each represented by a TreeNode.



# TreeView

- Important properties:
  - Nodes
  - SelectedNode
  - Sorted
  - CheckBoxes
  - ImageList

# TreeView

- Important Events:
  - AfterSelect
  - AfterExpand

# Exception Handling

# Common Exception Types

- System.ArgumentException - Thrown when a function is called with a bogus argument.

- System.ArgumentNullException - Subclass of ArgumentException that's thrown when a function argument is (unexpectedly) null.

- System.ArgumentOutOfRangeException - Subclass of ArgumentException that's thrown when a (usually numeric) argument is too big or too small. For example, this is thrown when passing a negative number into a function that accepts only positive values.

# Common Exception Types

- System.InvalidOperationException - Thrown when the state of an object is unsuitable for a method to successfully execute, regardless of any particular argument values. Examples include reading an unopened file or getting the next element from an enumerator where the underlying list has been modified partway through the iteration.

- System.NotSupportedException - Thrown to indicate that a particular functionality is not supported. A good example is calling the Add method on a collection for which IsReadOnly returns true.

- System.NotImplementedException - Thrown to indicate that a function has not yet been implemented.

# Secondary Forms

# Modal & Modeless

- Forms can exhibit either **modal** or **modeless** behavior
- A **modal form** is one that demands the user's immediate attention, and blocks input to any other windows the application may have open.
- Good UI design suggests that you should use **modal** dialogs **only** when absolutely necessary. Typical examples: error messages

# Modal & Modeless

```
//modal
Form frm = new Form(  );
frm.ShowDialog(  );

//modeless
Form frm = new Form(  );
frm.Show (  );
```

# UI Design

# Alignment