

Windows Forms

Szymon Szczepański szymon.szczepanski@gmail.com
Maciej Świechowski m.swiechowski@mini.pw.edu.pl
Paweł Aszklar p.aszklar@mini.pw.edu.pl

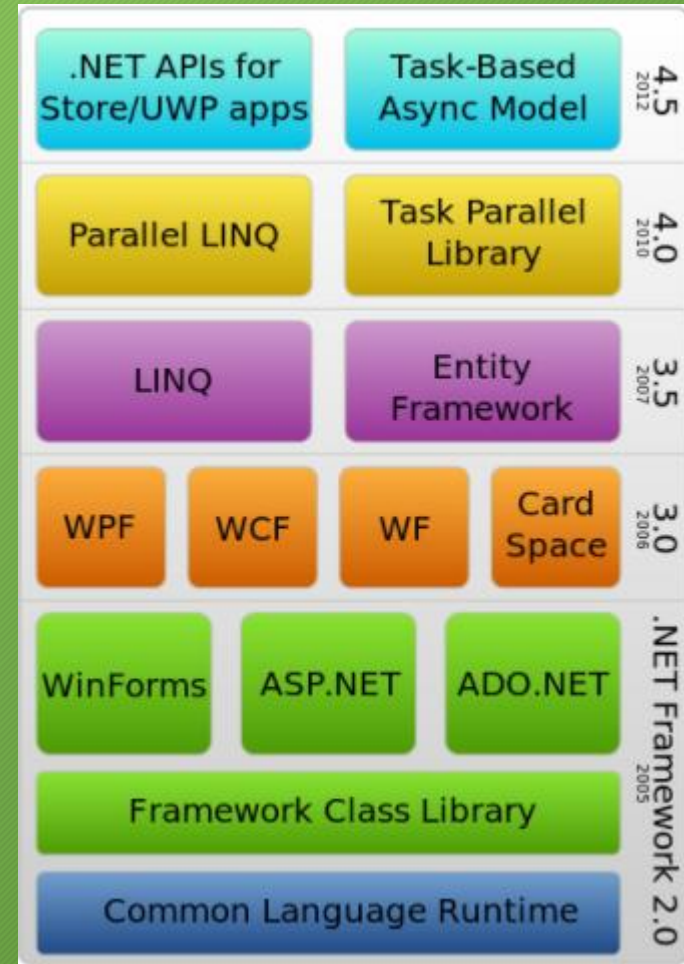
What is .NET Framework?

From 2002 to ...

- .NET 1.1 – 2003
- .NET 2.0 – 2005
- .NET 3.0 – 2006
- .NET 3.5 – 2007
- .NET 4.0 – 2010
- .NET 4.5 – 2012
- .NET 4.6 – 2015
- .NET 4.7 – 2017
- .NET 4.8 – 2019 (last major release)

For more:

https://en.wikipedia.org/wiki/.NET_Framework_version_history



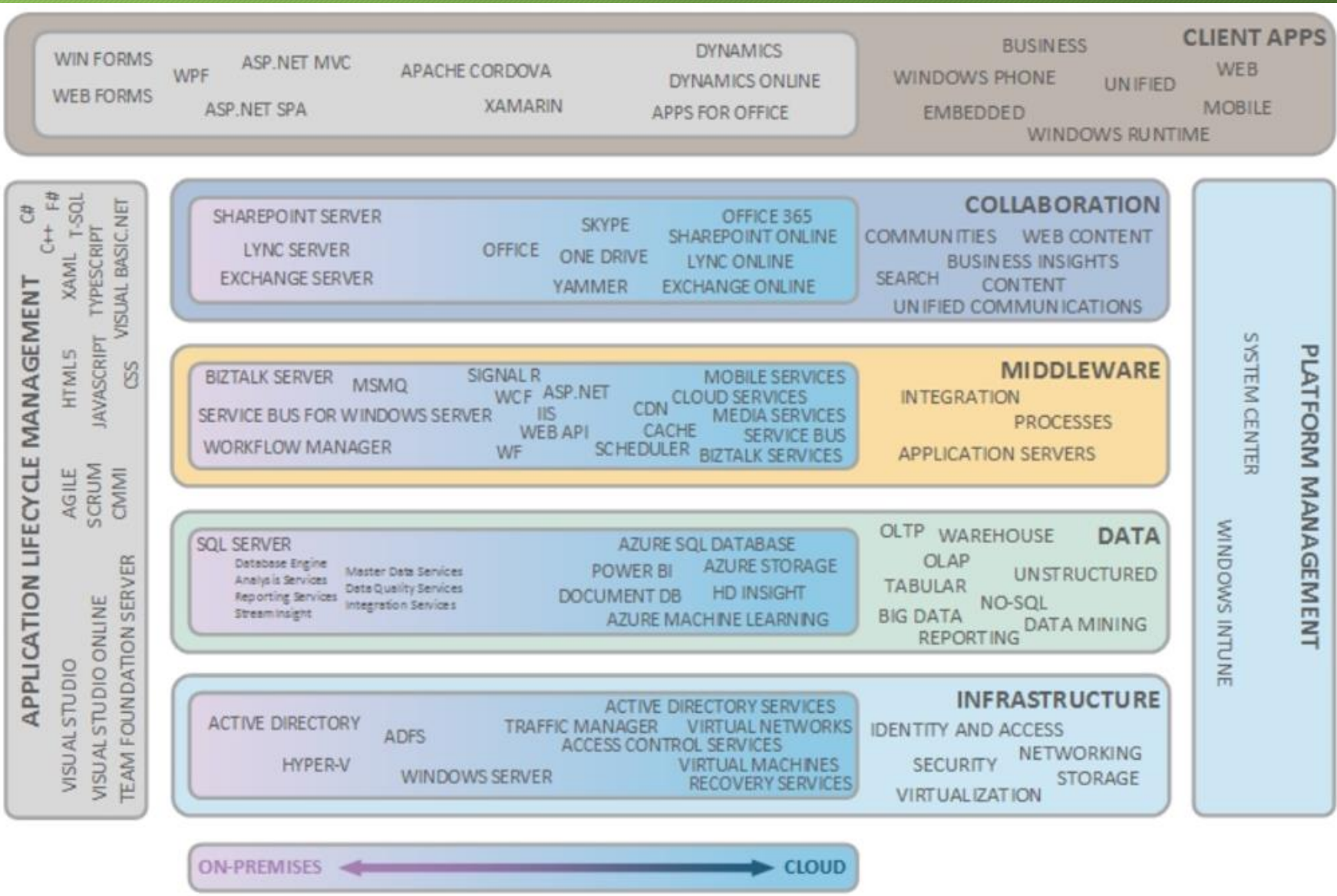
.NET Core

- Is NOT the .NET Framework
- .NET Core 1.0 – 2016
- .NET Core 2.0 – 2017
- .NET Core 3.0 – 2019
- .NET 5.0 – (future of .NET, 2020)
- .NET 6.0 – planned for 2021 (LTS)
- .NET 7.0 – planned for 2022
- .NET 8.0 – planned for 2023
- Windows Forms and WPF ported to .NET Core 3.1

.NET Core vs .NET Framework

- Use .NET Core for your server application when:
 - You have cross-platform needs.
 - You are targeting microservices.
 - You are using Docker containers.
 - You need high-performance and scalable systems.
 - You need side-by-side .NET versions per application.
- Use .NET Framework for your server application when:
 - Your app currently uses .NET Framework (recommendation is to extend instead of migrating).
 - Your app uses third-party .NET libraries or NuGet packages not available for .NET Core.
 - Your app uses .NET technologies that aren't available for .NET Core.
 - Your app uses a platform that doesn't support .NET Core.

Microsoft technology stack

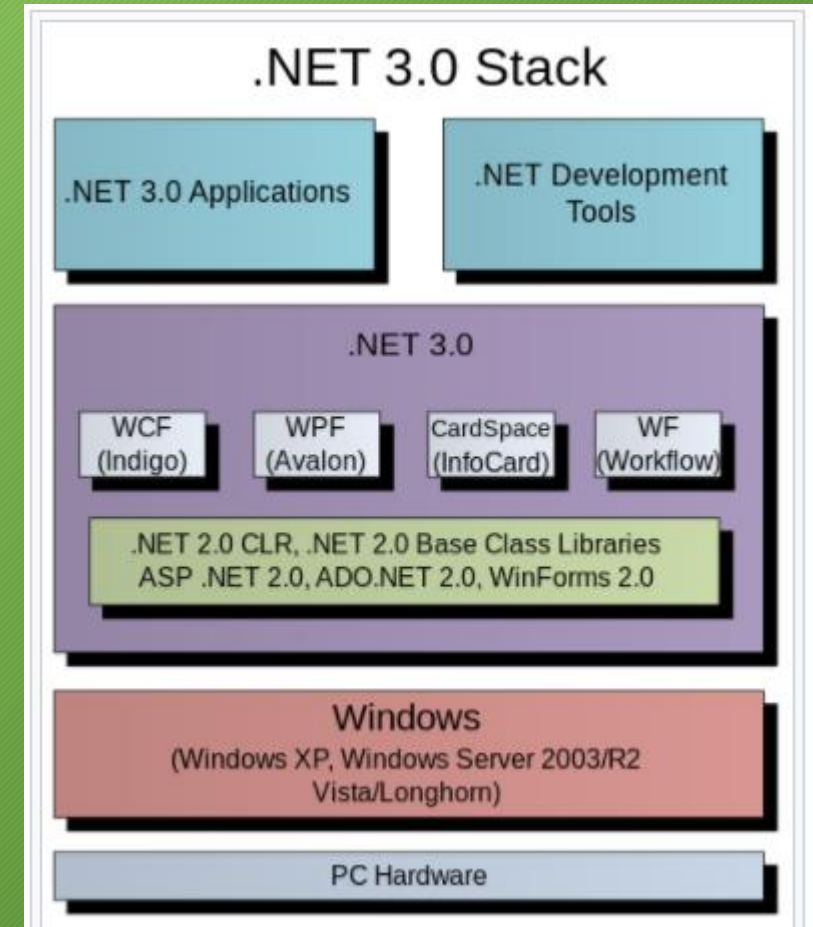


What is Windows Forms?

- Part of .NET Framework since version 1.0
- Created for Windows
- For Linux? MONO!

Namespace

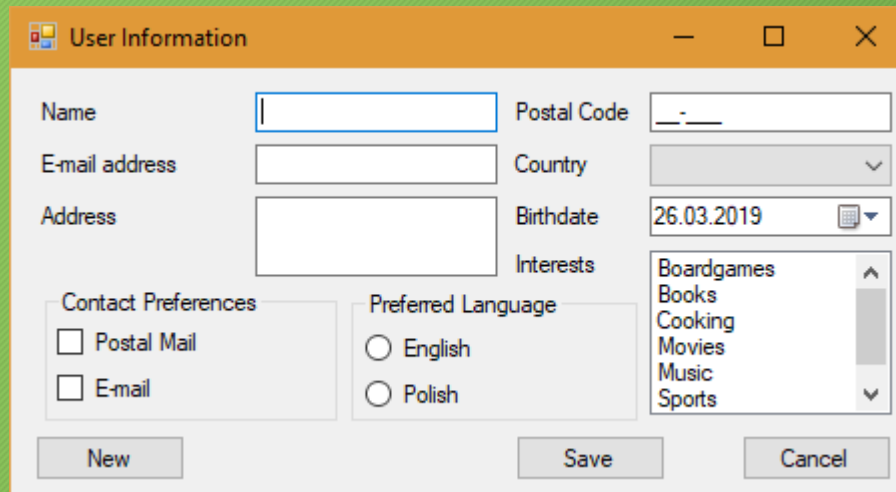
- System.Windows.Forms
- Abstraction over GUI part of Windows API exposing it to managed code
- Effectively a wrapper over part of Windows API
- Like .NET Framework it is *managed* but can have resource leaks



Categories of classes in Windows Forms

- Core infrastructure (e.g. ***Application, Forms***)
- Controls – derived from ***Control*** class (e.g. ***Button, TextBox***)
- Component – not derived from ***Control*** class (e.g. ***Timer, ToolTip***)
- Common dialog boxes (e.g. ***OpenFileDialog, PrintDialog***)

Some examples ...



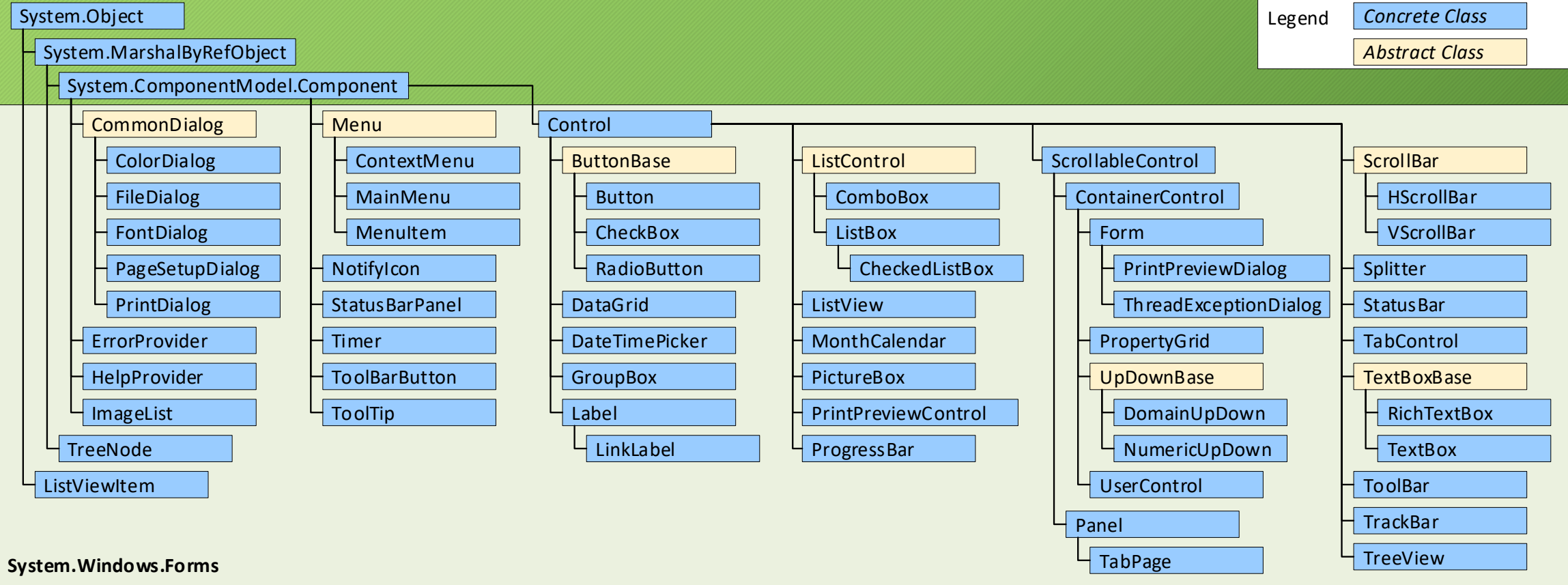
The image shows a screenshot of a Windows Forms dialog box titled "User Information". The dialog has an orange title bar with standard window controls (minimize, maximize, close). The main area contains several input fields and controls:

- Name:** A text box with a blue border.
- Postal Code:** A text box with a placeholder "____-____".
- E-mail address:** A text box.
- Country:** A dropdown menu.
- Address:** A text box.
- Birthdate:** A date picker showing "26.03.2019".
- Interests:** A list box containing "Boardgames", "Books", "Cooking", "Movies", "Music", and "Sports".
- Contact Preferences:** Two checkboxes: "Postal Mail" and "E-mail", both unchecked.
- Preferred Language:** Two radio buttons: "English" and "Polish", both unselected.

At the bottom of the dialog are three buttons: "New", "Save", and "Cancel".

Windows Forms - Architecture

- Event-driven applications
- Wrapping the existing Windows API in managed code
- More comprehensive abstraction above the Win32 API than Visual Basic or MFC
- Inheritance



Events and Events Handling

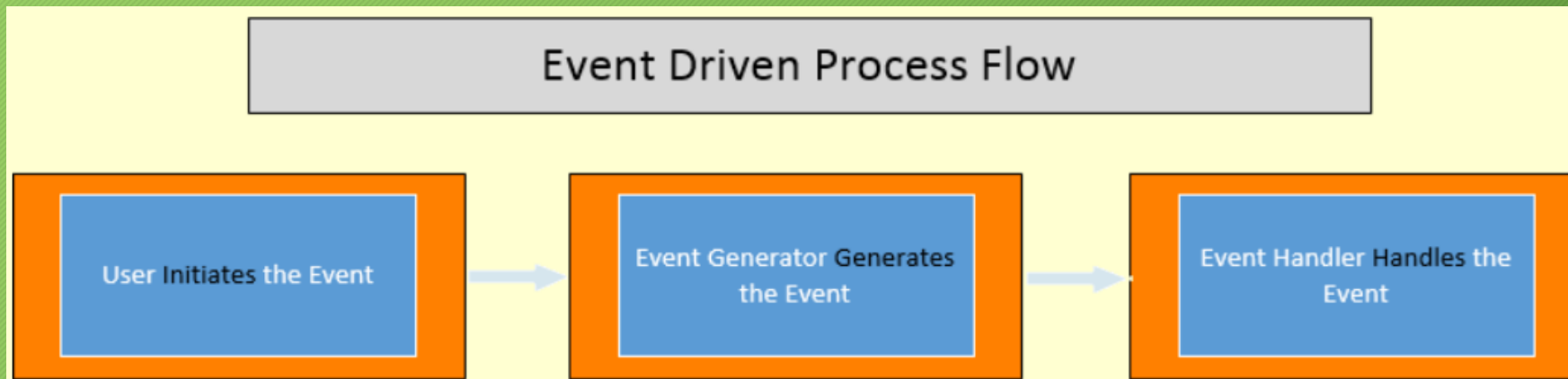
- Events enable a class or an object to notify other classes or objects when something of interest occurs. The class that sends (or raises) an event is called the publisher and the classes that receive (or handle) the event are called the subscribers.
- Event-driven graphical user interface
- Event handlers - methods that process events and perform tasks
- Each control generating an event has an associated delegate that defines the signature for event handlers
 - event delegates are multicast (they contain lists of methods' references)
 - once an event is raised, every method that the delegate references is called

Event-driven graphical user interface in Windows Forms

- In Windows Forms, event handler can be fired randomly
- In Windows Forms, events are by default synchronous
- Problems
 - Memory leaks (weak reference and GC)

Criticism:

The design of those programs which rely on event-action model has been criticized, and it has been suggested that the event-action model leads programmers to create error-prone, difficult to extend and excessively complex application code. Table-driven state machines have been advocated as a viable alternative. On the other hand, table-driven state machines themselves suffer from significant weaknesses including "state explosion" phenomena.



Events example

```
public class MyForm : Form
{
    public MyForm()
    {
        FormClosing += OnClosing;
    }

    private void OnClosing(Object sender, FormClosingEventArgs e)
    {
        if (MessageBox.Show("Sure to close?", "Question",
            MessageBoxButtons.YesNo) == DialogResult.No)
        {
            e.Cancel = true;
        }
    }
}
```


Application & Forms - let's start Windows Forms

Application (System.Windows.Forms)

- Class representing the entire Windows Forms application

MSDN :

The Application class has methods to start and stop applications and threads, and to process Windows messages, as follows:

- **Run** starts an application message loop on the current thread and, optionally, makes a form visible.
- **Exit** or **ExitThread** stops a message loop.
- **DoEvents** processes messages while your program is in a loop.
- **AddMessageFilter** adds a message filter to the application message pump to monitor Windows messages.
- **IMessageFilter** lets you stop an event from being raised or perform special operations before invoking an event handler.

This class has **CurrentCulture** and **CurrentInputLanguage** properties to get or set culture information for the current thread.

You cannot create an instance of this class.

Application

Properties with information about:

- path to the executable file
- path to application data directories
- current culture
- etc.

Forms (System.Windows.Forms) - Window

- Class representing the main window, dialog box, or MDI child window
- Most members inherited from parent classes
 - (most notably **Control** class)
- You will also need to add the STAThread attribute to the Main method in order for the form to run

```
System.Object
  System.MarshalByRefObject
    System.ComponentModel.Component
      System.Windows.Forms.Control
        System.Windows.Forms.ScrollableControl
          System.Windows.Forms.ContainerControl
            System.Windows.Forms.Form
```

Form's Lifetime

Windows Forms application starts, the startup events of the main form are raised in the following order:

1. **Constructor**
2. **Control.HandleCreated**
3. **Control.BindingContextChanged**
4. **Form.Load**
5. **Control.VisibleChanged**
6. **Form.Activated**
7. **Form.Shown**

When an application closes, the shutdown events of the main form are raised in the following order:

1. **Form.Closing**
2. **Form.FormClosing**
3. **Form.Closed**
4. **Form.FormClosed**
5. **Form.Deactivate**
6. **Dispose**
7. **Destructor**

Size and Postion

- Visibility
 - `Show()`, `Visible`
 - `Shown`, `VisibleChanged`
- Properties:
 - `Region`, `Bounds`, `DesktopBounds`, `ClientRectangle`
 - `Left`, `Top`, `Width`, `Height`
 - `Right == Left + Width`
 - `Bottom == Top + Height`
 - `StartPosition`
 - `Location`, `DesktopLocation`
 - `Size`, `ClientSize`
 - `MinimumSize`, `MaximumSize`
 - `AutoSize`, `AutoSizeMode`
 - `WindowState`, `TopMost`

Form's Size and Position cont'd

- **Methods:**

- `SetBounds()`, `SetDesktopBounds()`, `SetDesktopLocation()`
- `BringToFront()`, `SendToBack()`
- `SizeFromClientSize()`

- **Events:**

- `ClientSizeChanged`, `SizeChanged`
- `LocationChanged`
- `MaximumSizeChanged`, `MinimumSizeChanged`
- `Resize`, `ResizeBegin`, `ResizeEnd`

Form's Appearance

- Properties and methods for manipulating:
 - colors of different elements of the form
 - background images
 - fonts
 - icon, cursor icon
 - system buttons (minimize box, maximize box, help)
 - whether or not the form is visible on the taskbar
 - opacity of the form
 - and more...

Model and Modeless Forms

- A dialog (or dialogue) refers to a conversation between two people. In user interfaces, a dialog is a “conversation” between the system and the user, and often requests information or an action from the user.
- **Definition:** A **modal dialog** is a dialog that appears on top of the main content and moves the system into a special mode requiring user interaction. This dialog disables the main content until the user explicitly interacts with the modal dialog

Model and Modeless Forms

Modal

- Has to be closed before using the rest of the application
- ShowModal(), ShowDialog()
- AcceptButton, CancelButton, DialogResult

Modeless

- Allows to shift focus between different forms in the application
- Show()
- Close(), FormClosing, FormClosed

Guidelines for Using Modal Dialogs

- Use modal dialogs for important warnings, e.g. to prevent or correct critical errors.
 - Would the problem be easier or harder to correct if users' attention is taken away from the task?
 - Is the error irreversible?
- Use modal dialogs to request the user to enter information critical to continuing the current process
- Modal dialogs can be used to fragment a complex workflow into simpler steps.
- Use modal dialogs to ask for information that, when provided, could significantly lessen users' work or effort.
- Do not use modal dialogs for nonessential information that is not related to the current user flow.
- Avoid modal dialogs that interrupt high-stake processes such as checkout flows.
- Avoid modal dialogs for complex decision making that requires additional sources of information unavailable in the modal.