Windows Forms

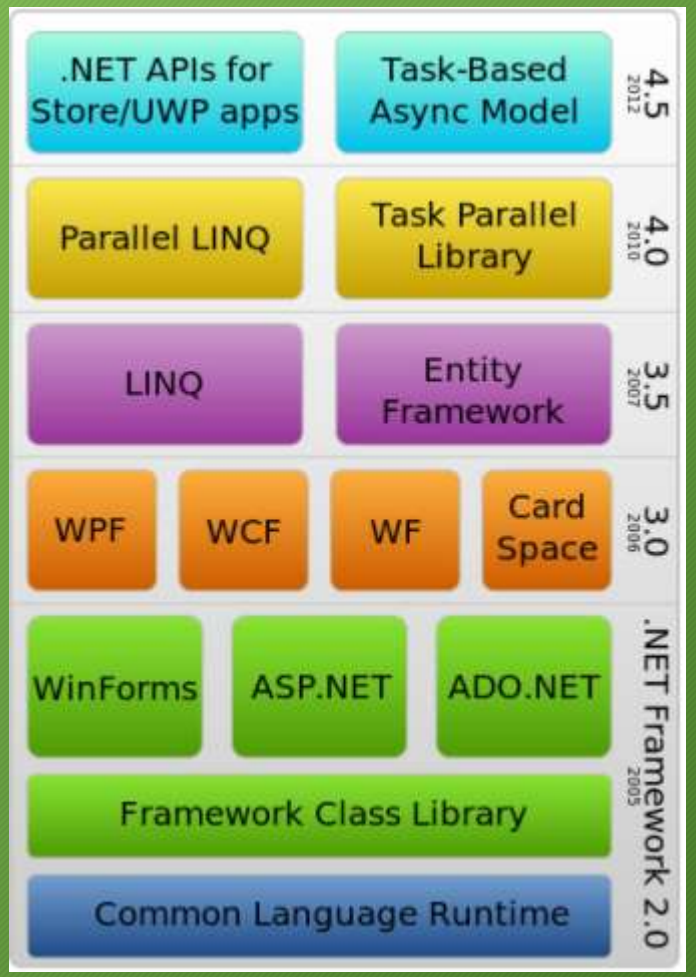
Szymon Szczepański szymon.szczepanski@gmail.com
Maciej Świechowski m.swiechowski@mini.pw.edu.pl
Paweł Aszklar p.aszklar@mini.pw.edu.pl

What is .NET Framework?

From 2002 to …

- .NET 1.0 – 2002
- .NET 2.0 – 2005 (64-bit support, partial classes, generics)
- .NET 3.0 – 2006
- .NET 3.5 – 2007
- .NET 4.0 – 2010
- .NET 4.5 – 2012
- .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
- Windows Forms and WPF ported to .NET Core 3.1
- Older, Unsupported:
  - .NET Core 1.0 – 2016
  - .NET Core 3.1 – 2019
  - .NET 5.0 – („future of .NET”, 2020)
- .NET 6.0 – 2021 (LTS)
- .NET 7.0 – 2022
- .NET 8.0 – 2023 (current, LTS)
- .NET 9.0 – planned for November 2024
- .NET 10.0 – planned for 2025 (next LTS)

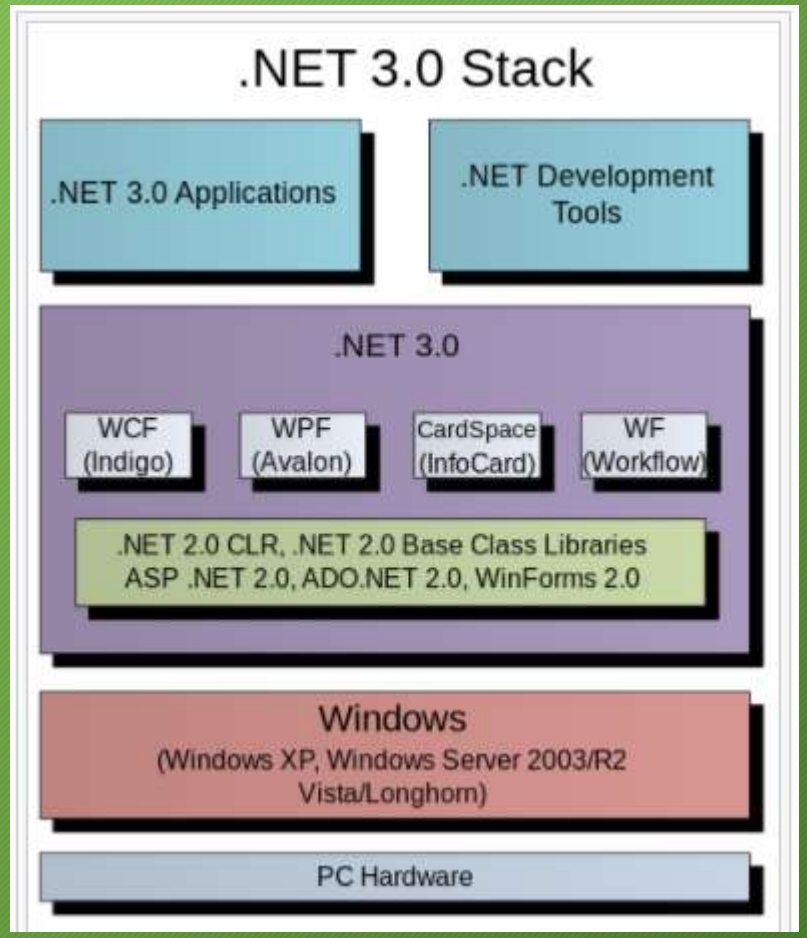
What is Windows Forms?

- Part of .NET Framework since version 1.0
- Created for Windows
- Other platforms? Mono – but it has issues…

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 all of .NET it is *managed* but can have resource leaks

Open Source in .NET https://github.com/dotnet/winforms



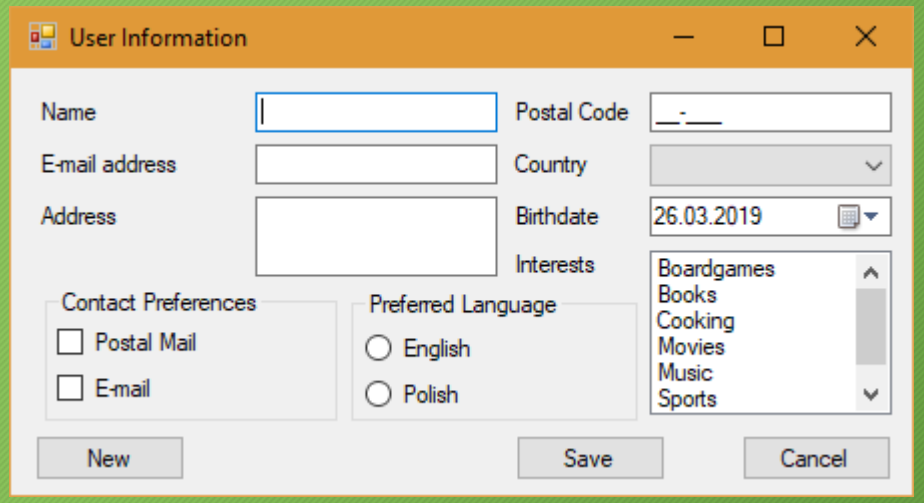
Resources

- Prerequisites:
  - Solid C# and .NET foundations
  - Understanding of WinAPI inner workings
- .NET Docs (under construction):
  https://learn.microsoft.com/en-us/dotnet/desktop/winforms/get-started/create-app-visual-studio?view=netdesktop-8.0
- .NET Framework Docs:
  https://learn.microsoft.com/en-us/dotnet/desktop/winforms/windows-forms-overview?view=netframeworkdesktop-4.8
- For more structured introduction:
  Any solid C# book ca. 2002-2008
- Books focusing on Windows Forms exist, but it’s hard to recommend any (mostly rethreading of documentation)

Categories of classes in Windows Forms

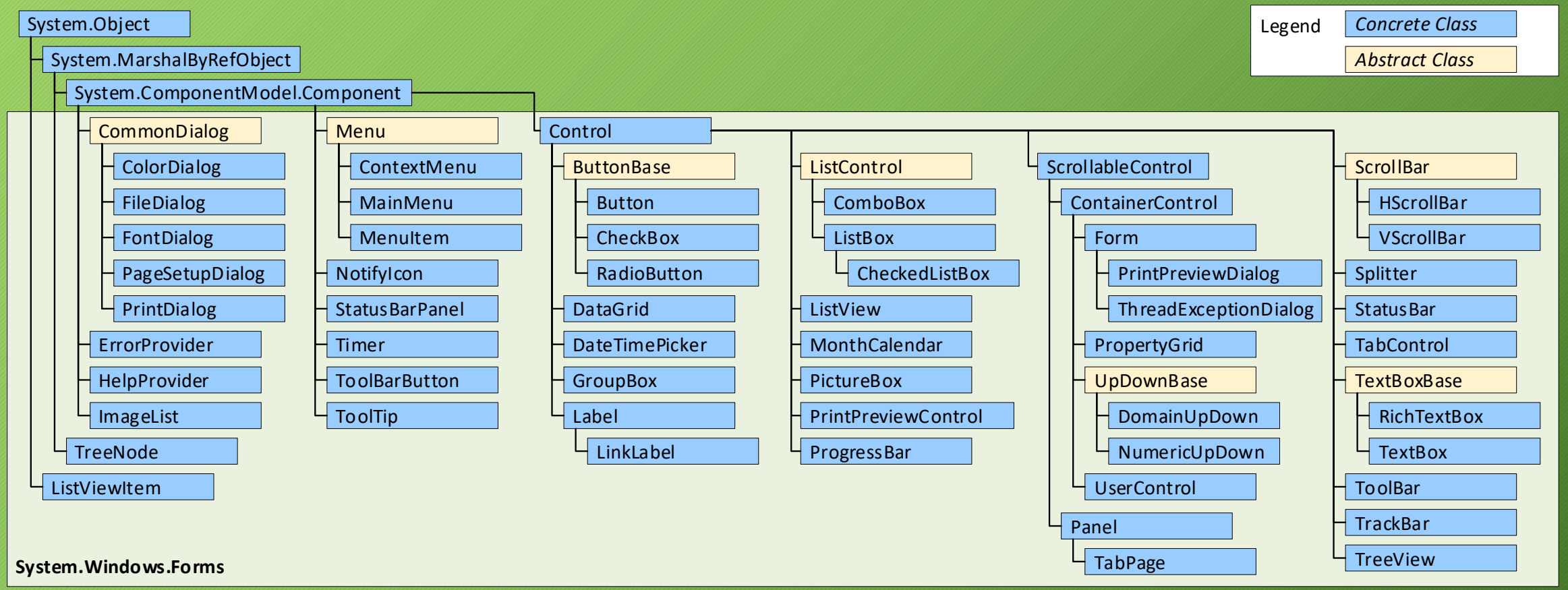
- Core infrastructure (e.g. ***Application***, ***Forms***)
- Controls – derived from ***Control*** class (e.g. ***Button***, ***TextBox***)
- Component – not derived from ***Control*** class (e.g. ***Timer***, ***ToolTip***)
- Common dialog boxes (e.g. ***OpenFileDialog***, ***PrintDialog***)

Some examples …



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

- Class or object can notify other classes or objects when something of interest occurs.
- Class or object sending (or raising) an event is called the publisher (sender)
- Classes or objects that receive (or handle) the event are called the subscribers.
- GUI in Windows Forms is Event-driven
- Windows Forms events roughly equivalent to messages in WinAPI
- Event handlers – methods (delegates) that process events and perform tasks
  - Same delegate can be used for many events
- Each control generating an event has an associated method signature for event handlers
  - events are multicast (they contain lists of referenced methods)
  - once an event is raised, every referenced method is called
  - invocation order is not specified!

Event-driven graphical user interface in Windows Forms

- In Windows Forms, order of events and associated handlers can be hard to determine
- In Windows Forms, events are by default synchronous, handlers executed on UI thread
- Problems
  - Memory leaks (weak reference and GC)

Criticism:

- Can lead to error-prone, difficult to extend and excessively complex application code.
- Alternative: table-driven state machines – suffers from "state explosion".

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.

It’s a static class, 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)
- It is highly recommended to add the STAThread attribute to the Main method

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

When a Form is created and shown (e.g. on application startup), events 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 a Form is destroyed (e.g. when application's main form is closed by the user) events are raised in the following order:

1. **Form.Closing [obsolete, not triggered on Application.Exit()]**
2. **Form.FormClosing (cancelable)**
3. **Form.Closed [obsolete, not triggered on Application.Exit()]**
4. **Form.FormClosed**
5. **Form.Deactivate**
6. **Dispose**
7. **Destructor**

Size and Postion

- Visibility
  - **Show(), Hide(), Visible**
  - **Shown, VisibleChanged**
- Properties:
  - **StartPosition: Manual, CenterScreen, CenterParent, etc.**
  - **Bounds, Location, Size [Screen Coordinates]**
  - **Left, Top, Width, Height [Screen Coordinates]**
    - **Right == Left + Width**
    - **Bottom == Top + HeightDesktopLocation,**
  - **DesktopBounds [Workspace Coordinates]**
  - **ClientSize, ClientRectangle**
  - **Region**
  - **MinimumSize, MaximumSize**
  - **AutoSize, AutoSizeMode (GrowOnly, GrowAndShrink)**
  - **WindowState (Minimized, Maximized, Normal), TopMost**

Form’s Size and Position cont’d

- Methods:
  - `SetBounds(), SetDesktopBounds(),SetDesktopLocation()`
  - `BringToFront(), SendToBack()`
  - `SizeFromClientSize()`
- Events:
  - `Layout`
  - `Move, LocationChanged`
  - `Resize, SizeChanged, ClientSizeChanged`
  - `ResizeBegin, ResizeEnd`
  - `MaximumSizeChanged, MinimumSizeChanged`

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…

Modal 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

Modal 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

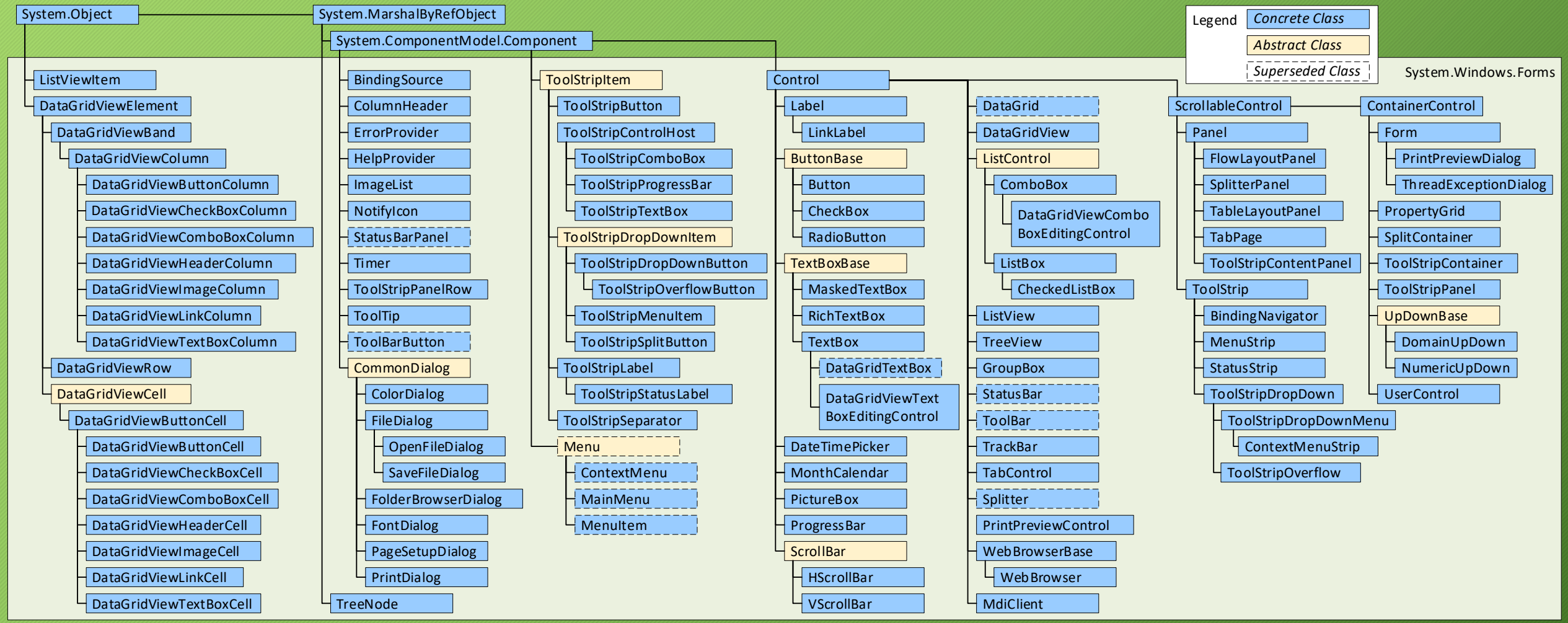
Guidelines for Using Modal Dialogs

- 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.

Controls – Base classes

- **`Component`** - base class for (almost) all classes within the **`System.Windows.Forms`** namespace
- **`Control`** - message routing, keyboard and mouse, security, size and position, HWND
  - **`Controls`** property - collection of child controls
- **`ScrollableControl`** - auto-scrolling
- **`ContainerControl`** - hosting other controls, focus, tab order
- **`UserControl`** - composite control consisting of one or more controls

Control Classes Hierarchy



Control class

- Size and location
  - mostly the same as in **Form**
    - actually, the other way round
- Auto-placement and auto-resize
  - **`Anchor`** – position relative to the edge of its container
  - **`Dock`**
- Z-order
  - **`BringToFront()`**, **`SendToBack()`**
- Styles:
  - **`SetStyle()`**, **`GetStyle()`**, **`UpdateStyles()`**
- **`Tag`** – allows associating custom data (of any type) with the control
- Tabbing
  - **`TabStop`**
  - **`TabIndex`**
- Focus
  - **`ControlStyles.Selectable`**

Control Class cont'd

- Parent-child relationships:
  - **`Controls`** – collection of all child controls
  - **`HasChildren –`** returns **`true`** if the control has child controls
  - **`Parent –`** the control object that contains this control (may be **null**)
  - **`TopLevelControl –`** control at the very top of the hierarchy
- Ambient properties
  - using parent's values if not set
  - e.g. **`BackColor`**
- To apply operating system visual theme to control
  - **`Application.EnableVisualStyles()`**
- Accessibility support
  - Aim: application available for users with disabilities
  - **`AccessibleXXX`**