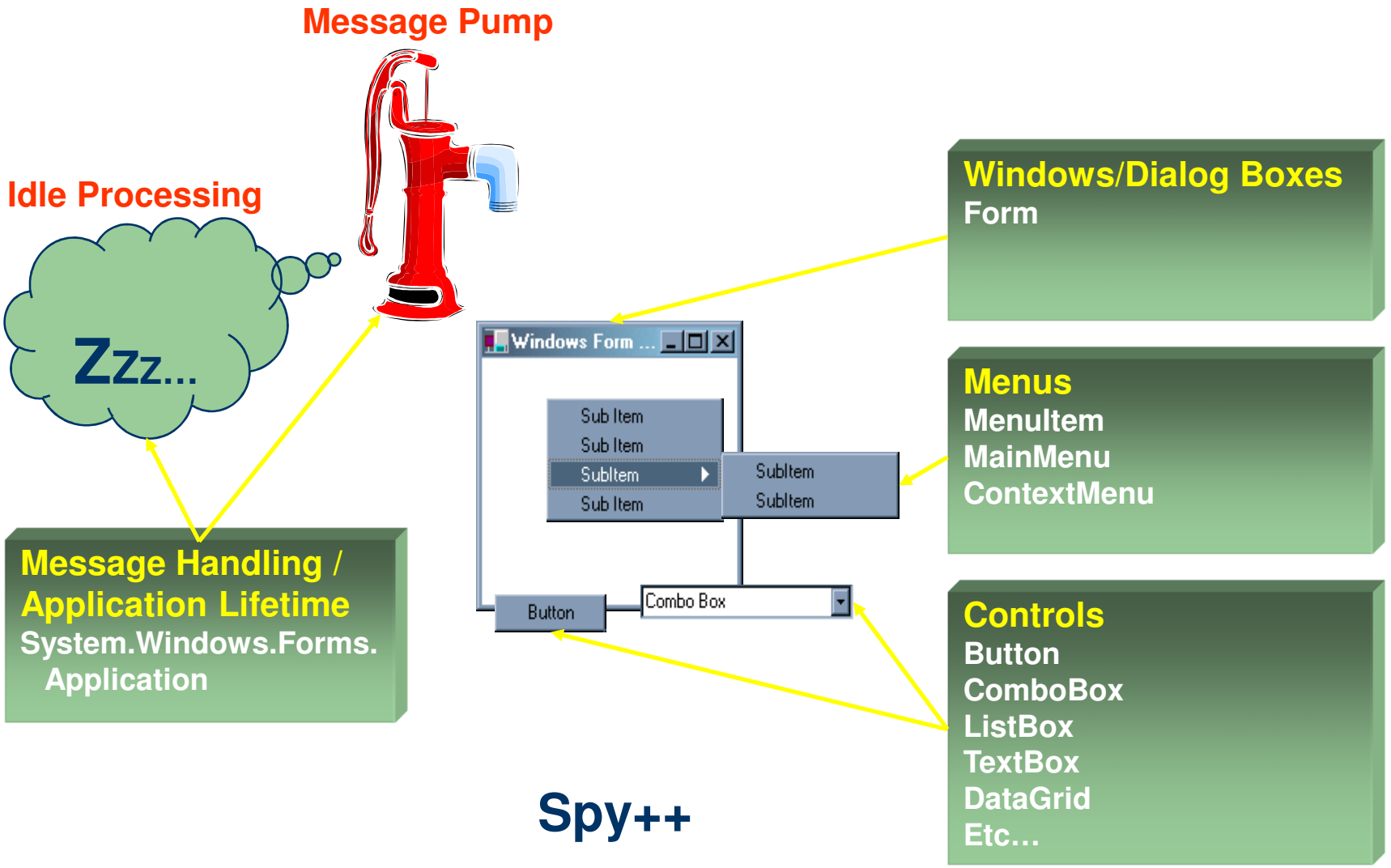
Ingegneria del Software T

Interfaccia utente

Creating Windows Applications

- Typical windows-application design & development
  - 1+ classes derived from `System.Windows.Forms.Form`
  - Design UI with VisualStudio .NET
  - Possible to do anything directly via code
- Typical windows-application threading
  - A single thread dedicated to UI
    - Runs the message pump
    - Can do other things, but blocks only briefly (or never)
  - Background threads used for lengthy non-UI functionality

Elements of a Windows Application

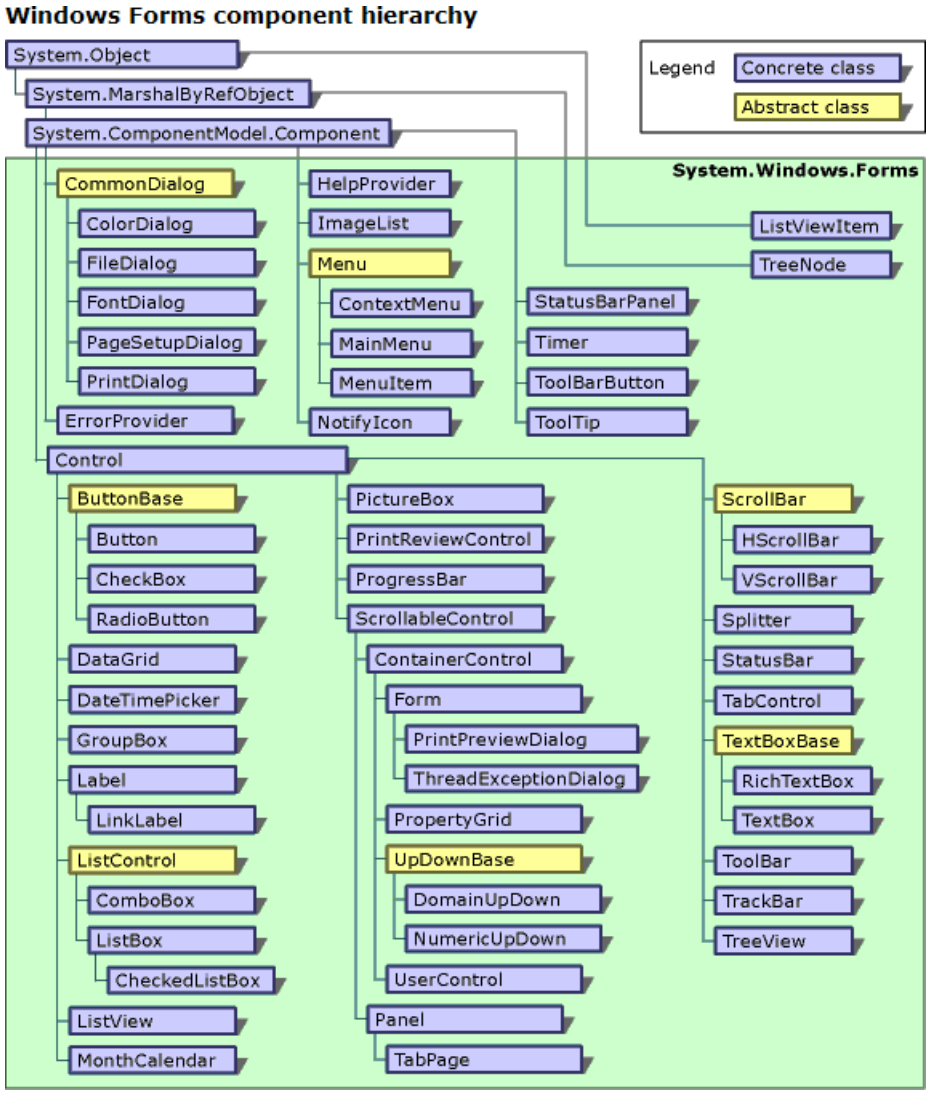


`System.Windows.Forms.Application` class

- Non-instantiable class with public static methods, properties and events
- Used to handle windows-application infrastructure
  - Message pump methods
    - **`Run(Form form)`**
    - **`Exit()`** - Informs all message pumps that they must terminate, and then closes all application forms after the messages have been processed
  - Application level events
    - **`Idle`**, **`ApplicationExit`**

`System.Windows.Forms` namespace

- Contains classes for creating Windows-based applications
- The classes can be grouped into the following categories:
  - **Components**
  - **Common Dialog Boxes**
  - **Controls**
    - **Form**
    - **UserControl**



`System.Drawing` namespace

- Contains basic graphic objects
  - Classes: **`Graphics`**, **`Font`**, **`Brush`**, **`Pen`**, **`Icon`**, **`Bitmap`**, ...
  - Instance creators: **`Brushes`**, **`Pens`**, **`SystemBrushes`**, **`SystemColors`**, **`SystemIcons`**, **`Cursors`**
  - Structures: **`Point`**, **`Size`**, **`Rectangle`**, **`Color`**, ...
- **`System.Drawing.Graphics`**
  - Represents a **drawing surface**
  - Can be in-memory, form-based, or HDC-based
  - Used to draw and paint on controls
    - **`DrawString()`**, **`DrawImage()`**, **`FillEllipse()`**, **`FillRectangle()`**, ...

Components

- The term **component** is generally used for an object that is **reusable** and can interact with other objects
- A .NET Framework `Component` satisfies those general requirements and additionally provides **design-time support**
- A component can be used in a rapid application development (RAD) environment
  - can be added to the toolbox of Visual Studio .NET
  - can be dragged and dropped onto a form
  - can be manipulated on a design surface
- Base design-time support is built into the .NET Framework
  ► a component developer does not have to do any additional work to take advantage of the base design-time functionality

Esempio 4.0.1

Common Dialog Boxes

- Common dialog boxes can be used to give your application a consistent user interface when performing tasks such as opening and saving files, manipulating the font or text color, or printing
  - **`OpenFileDialog`** and **`SaveFileDialog`**
  - **`FontDialog`**
  - **`ColorDialog`**
  - **`PageSetupDialog`**, **`PrintPreviewDialog`**, and **`PrintDialog`**
- In addition, the **`System.Windows.Forms`** namespace provides the **`MessageBox`** class for displaying a message box that can display and retrieve data from the user

Esempio 4.0.2

Controls

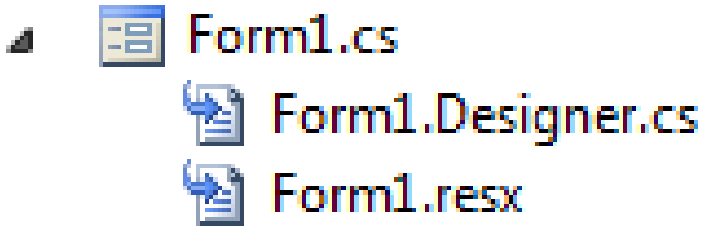
- A control is a component that provides (or enables) user-interface (UI) capabilities
- Some controls are designed for **data entry**
  - **`TextBox`**, **`ComboBox`**, …
- Other controls **display application data**
  - **`Label`**, **`ListView`**, **`TreeView`**, **`DataGridView`**, …
- The namespace also provides controls for **invoking commands** within the application
  - **`Button`**, **`LinkLabel`**, …
- **Containers** of child controls
  - **`Panel`**, **`SplitContainer`**, **`TableLayoutPanel`**, …
- **Containers** of components
  - **`ToolStrip`**, **`MenuStrip`**, **`ContextMenuStrip`**, …

`System.Windows.Forms.Control` class

- Base-class for all controls/forms
  - Derives from **`Component`**
  - Provides the base functionality for all controls
  - Wraps an underlying **OS window handle**
- Implements many
  - **Properties** for modifying settings of an instance
    - **`Size`**, **`BackColor`**, **`ContextMenu`**, ...
  - **Methods** for performing actions on an instance
    - **`Show()`**, **`Hide()`**, **`Invalidate()`**, ...
  - **Events** for "external" registration for event notification
    - **`Click`**, **`DragDrop`**, **`ControlAdded`**, …
- Derived classes override and specialize functionality
  - Specialized methods, properties, and events
    - **`TextBox`** – **`PasswordChar`**, **`Undo()`**, **`Copy()`**
    - **`Button`** – **`Image`**, **`PerformClick()`**

Using Controls
(by designer)

1. Add the control to the container
2. Affect the control **appearance** and **behavior** by setting **properties**
3. Define and register methods to **handle GUI events**
   - Buttons clicks, menu selections, mouse movements, timer events, etc.
   - Default behavior implemented by base classes



Esempio 4.0.3

Using Controls (by code)

1. Create and add the control

```
Button button = new Button();
container.Controls.Add(button);
```

2. Set properties

```
button.Text = "A Button";                // set text
button.Location = new Point(10, 10);     // and location
```

3. Define event handler

```
private void ButtonClicked(object sender, EventArgs e)
{
  MessageBox.Show("The button was clicked!");
}
```

4. Register for event notification

```
button.Click += ButtonClicked;
```

`System.Windows.Forms.Form` class

- A specialized derivation of **`Control`** used to implement a **top-level window** or **dialog**
- Gains much of its functionality from base classes
- Specialized to
  - Contain a title-bar, system menu, minimize/maximize
  - Contain a main menu
  - Manage dialog buttons
  - Implement MDI - Multiple Document Interface
  - ...
- Your applications derive from **`Form`** to create
  - Windows
  - Dialog boxes

Using Forms

- Create a Form-derived class

```
class BlueForm : Form
{
  public BlueForm()
  { BackColor = Color.Blue; }
}
```

1. Start message loop and display form

```
Application.Run(new BlueForm());
```

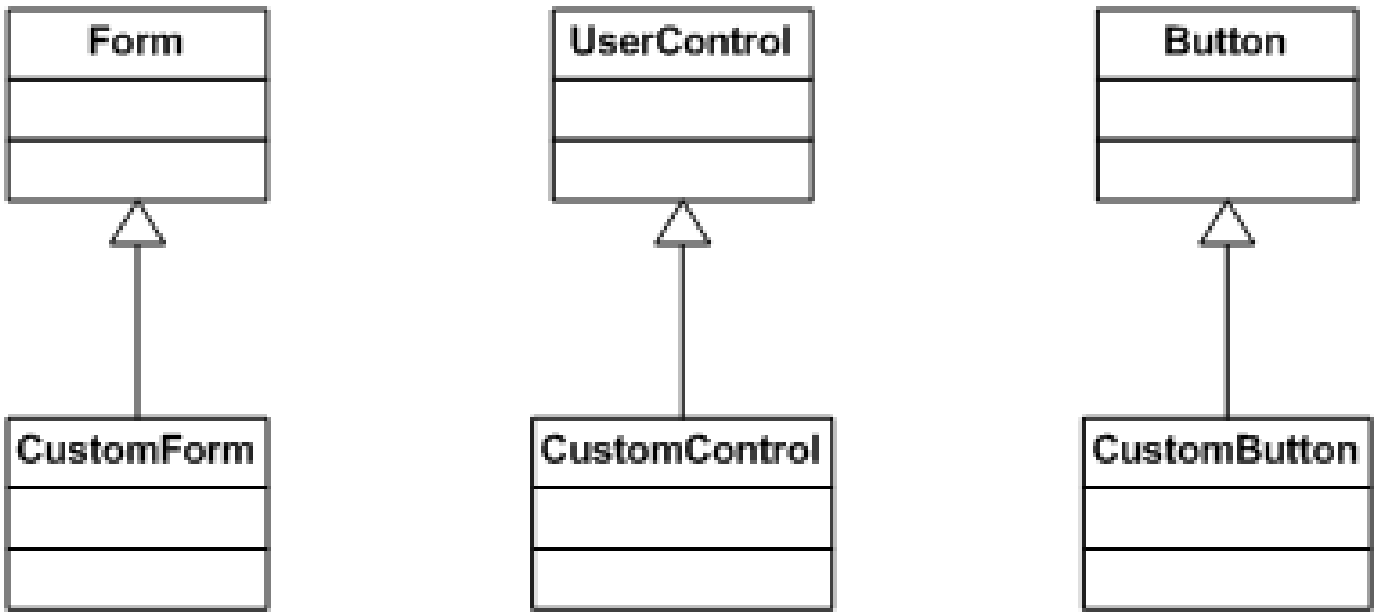
2. Show the derived form (modeless)

```
Form form = new BlueForm();   // Display on current
form.Show();                  // thread's message loop
```

3. Show the derived form as a dialog (modal)

```
Form form = new BlueForm();   // Display on current
form.ShowDialog();            // thread's message loop
```

Custom controls



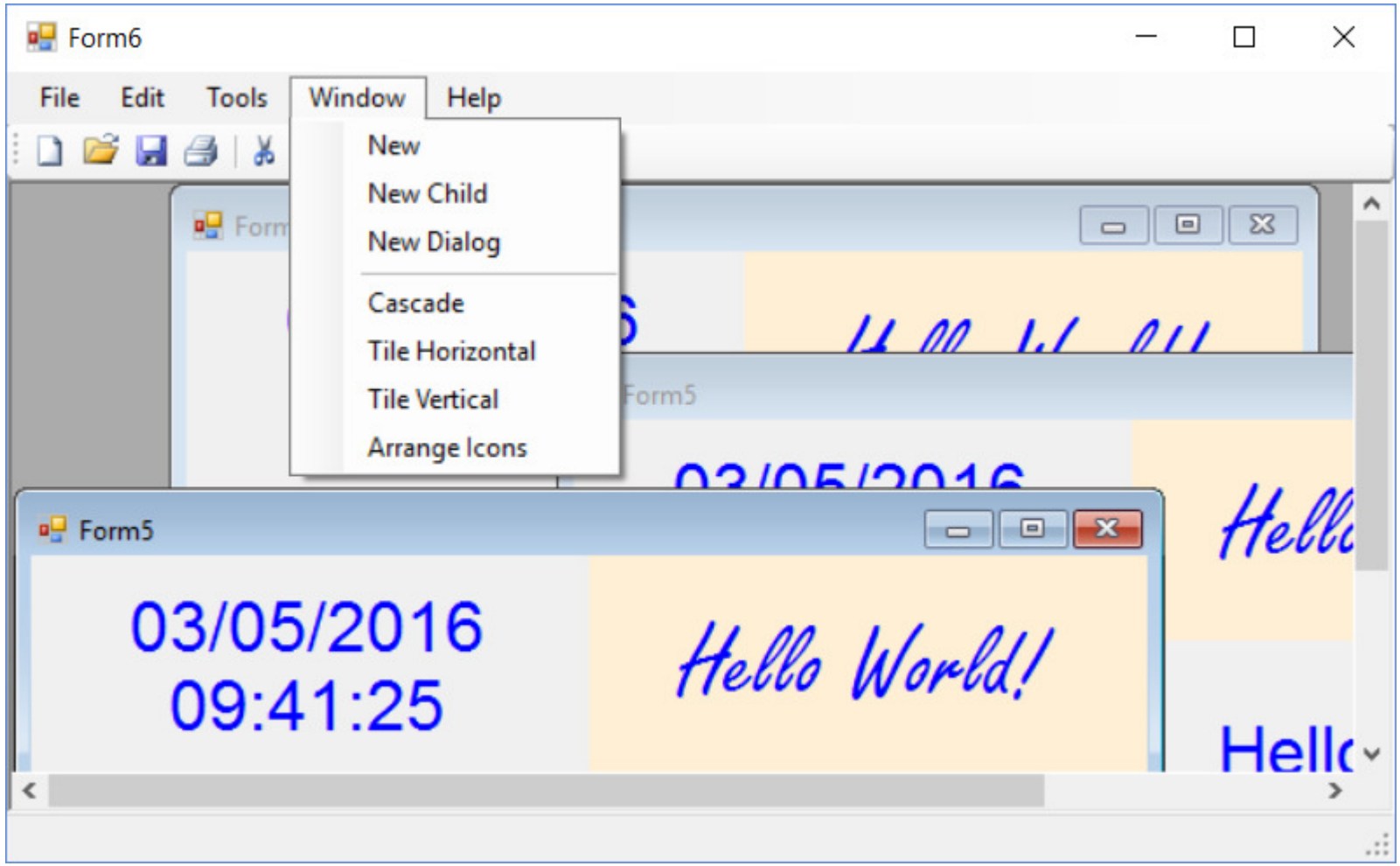
Custom controls

- Override virtual methods for handling GUI
  - `OnPaint()`, `OnMouseMove()`, `OnLoad()`, `OnFormClosing()`, ...
  - Do not override when default functionality is ok (usually the case)
  - When overriding a virtual method, call the base-implementation of the method

```
protected override void OnPaint(PaintEventArgs e)
{
  base.OnPaint(e);
  // Do some work
}
```

Esempio 4.0.4 + 4.1-5

Multiple Document Interface



Multiple Document Interface

- Nel costruttore della **`MainForm`**:

```
IsMdiContainer = true;
```

- Per aggiungere una **`ChildForm`**:

```
Form childForm = new ChildForm();
childForm.MdiParent = mainForm;
childForm.Show();
```

Esempio 4.6

Components

- The **`System.Windows.Forms`** namespace provides classes that do not derive from the **`Control`** class but still provide visual features to a Windows-based application
- The **`ToolTip`** and **`ErrorProvider`** classes provide information to the user
- The **`Help`** and **`HelpProvider`** classes enable you to display help information to the user of your applications

Esempio 4.7