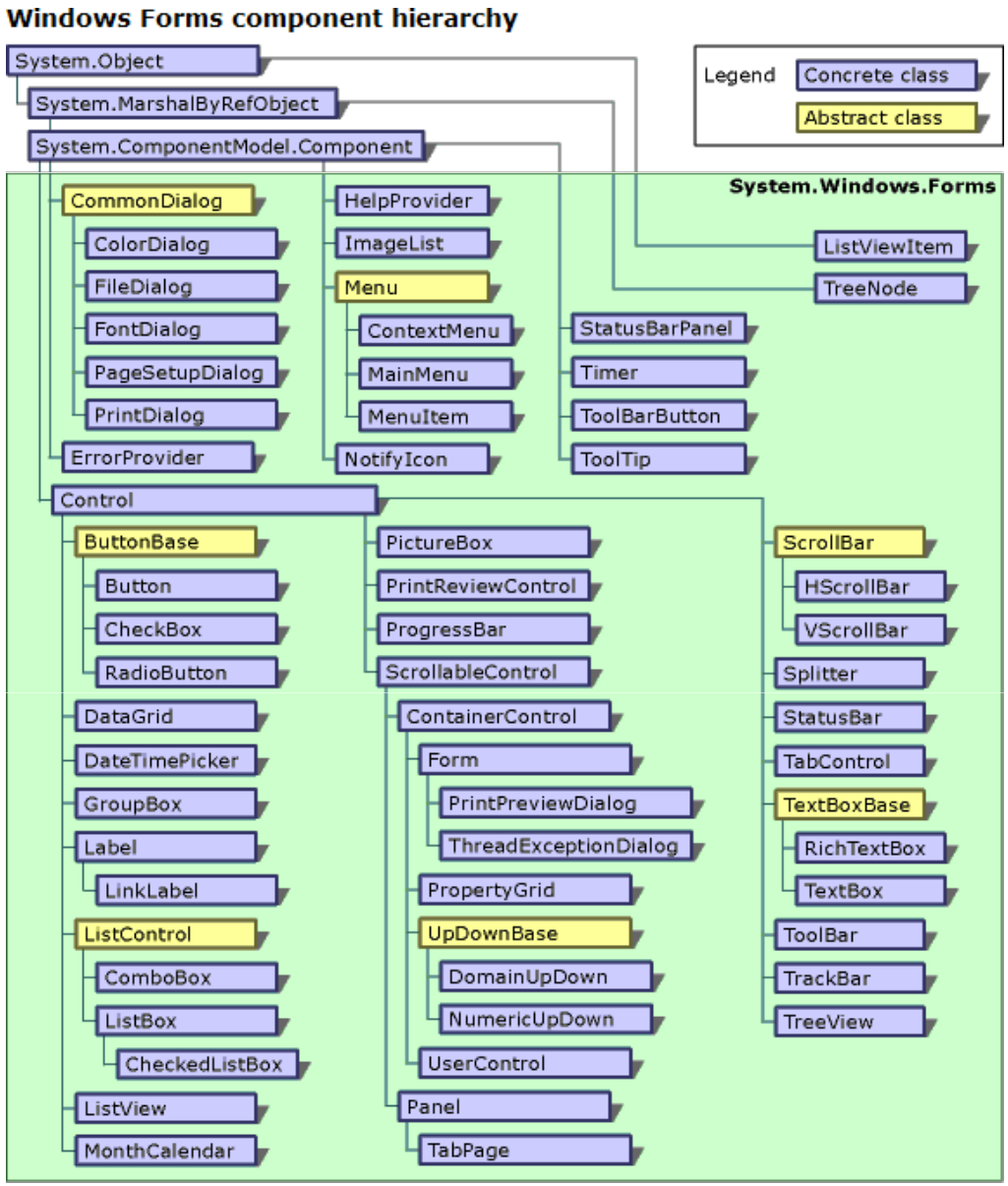
Laboratorio di Ingegneria del Software L-A

Interfaccia utente

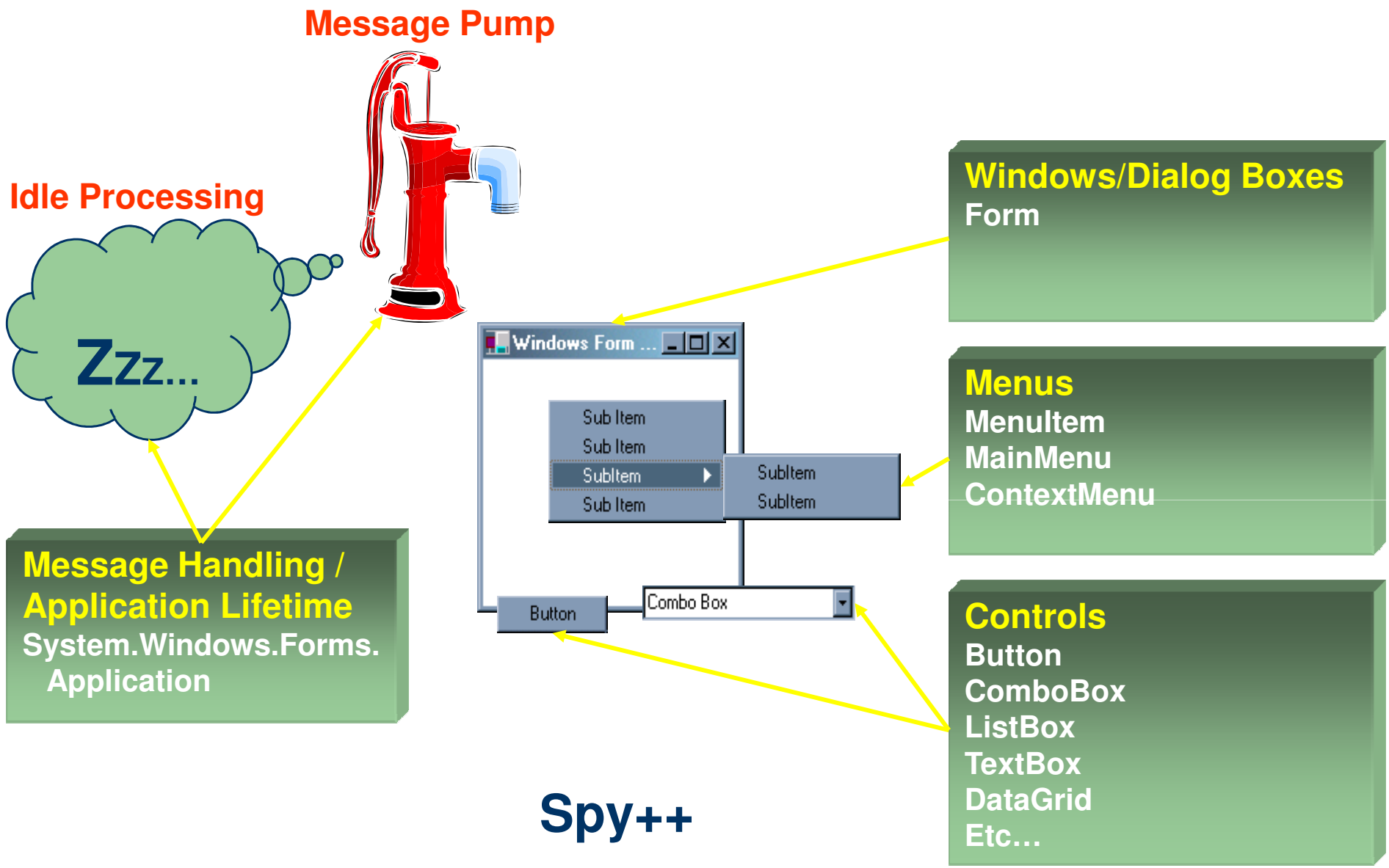
`System.Windows.Forms`

- The **`System.Windows.Forms`** namespace contains classes for creating Windows-based applications
- The classes can be grouped into the following categories:
  - **Form**, **Control**, and **UserControl**
  - **Controls**
  - **Components**
  - **Common Dialog Boxes**

System.Windows.Forms



Elements of a Windows Application



HelloWorld Windows Application

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

namespace HelloWorld
{
  static class Program
  {
    [STAThread]  //  COM threading model
    static void Main()
    {
      Application.EnableVisualStyles();
      Application.SetCompatibleTextRenderingDefault(false);
      Application.Run(new HelloWorldForm());
    }
  }
}
```

HelloWorld Windows Application

```
namespace HelloWorld
{
  public partial class HelloWorldForm : Form
  {
    public HelloWorldForm()
    {
      InitializeComponent();
    }

    protected override void OnPaint(PaintEventArgs e)
    {
      base.OnPaint(e);
      e.Graphics.DrawString("Hello World!",
        new Font("Arial",36), Brushes.Blue, 10, 100);
    }
  }
}
```

Sostituire con `Label`

Creating Windows Applications

- Typical windows-application design
  - One or more classes derived from **`System.Windows.Form`**
- Derived classes
  - Affect instance appearance and behavior by setting **properties**
  - Create objects to **implement GUI controls**
    - Buttons, text boxes, menus, timers, custom controls, etc.
  - Add controls to their UI
  - Implement methods to **handle GUI events**
    - Buttons clicks, menu selections, mouse movements, timer events, etc.
    - Default behavior implemented by base classes

Creating Windows Applications

- 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
- Typical windows-applications development
  - Design UI with VisualStudio .NET
    - Possible to do anything directly via code
  - Also use classes in **`System.Drawing`** namespace

`System.Drawing` namespace

- Full of types used heavily in windows applications
- Implements 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`**
  - Important class that represents a **drawing surface**
  - Can be in-memory, form-based, or HDC-based
  - Used by forms applications to draw and paint on controls
    - **`DrawString()`**, **`DrawImage()`**, **`FillEllipse()`**, **`FillRectangle()`**, ...

`System.Windows.Forms.Application`

- Non-instantiable class with public static methods and properties
- 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`**

Controls

- A control is a component that provides (or enables) user-interface (UI) capabilities
- The .NET Framework provides two base classes for controls:
  - **`System.Windows.Forms.Control`**
    - for client-side Windows Forms controls
  - **`System.Web.UI.Control`**
    - for ASP.NET server controls
- All controls in the .NET Framework class library derive directly or indirectly from these two classes

Controls

- The **`System.Windows.Forms`** namespace provides a variety of control classes that allow you to create rich user interfaces
- Some controls are designed for **data entry**
  - **`TextBox`**, **`ComboBox`**, …
- Other controls **display application data**
  - **`Label`**, **`ListView`**, …
- The namespace also provides controls for **invoking commands** within the application
  - **`Button`**, **`ToolBar`**, …

`System.Windows.Forms.Control`

- Base-class for all controls/forms in managed code
  - Provides the base functionality for all controls that are displayed on a **`Form`**
  - Derives from **`Component`**
  - 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`**, ...
- Instances of **`Control`** can contain child controls

`System.Windows.Forms.Control`

- Derived classes override and specialize functionality
  - Specialized methods, properties, and events
    - **`TextBox`** – **`PasswordChar`**, **`Undo()`**, **`Copy()`**
    - **`Button`** – **`Image`**, **`PerformClick()`**
  - The **`Form`** class is derived from **`Control`**
- To create a **custom control** that is a composite of other controls, use the **`UserControl`** class

`System.Windows.Forms.Form`

- 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 main menu
  - Contain a title-bar, system menu, minimize/maximize
  - Implement MDI - Multiple Document Interface
  - Manage dialog buttons
  - ...
- 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
```

Using Forms

- In the type's constructor
  - Set properties
  - Create child controls
    - Use the **`Controls`** property to add controls to the form
  - Setup the form's menu
- Override virtual methods for handling GUI
  - **`OnClosing()`**, **`OnPaint()`**, **`OnMouseMove()`**, ...
  - Do not override when default functionality is ok (usually the case)
  - When overriding a virtual method, usually call the base-implementation of the method

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

Multiple Document Interface

- Nel costruttore della MainForm:
  - `IsMdiContainer = true;`
- Per aggiungere una ChildForm:
  - ```
    Form childForm = new ChildForm();
    childForm.MdiParent = mainForm;
    childForm.Show();
    ```
    ```

Using Controls

- Create the control

```
Button ctrl = new Button();  // Create a button
```

- Set properties

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

- Add the control to your forms Controls collection

```
myForm.Controls.Add(ctrl); // Add the control to form
```

- Define event handler

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

- Register for event notification

```
// Register ButtonClicked as an event handler
ctrl.Click += new EventHandler(ButtonClicked);
```

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
  - The **`OpenFileDialog`** and **`SaveFileDialog`** classes provide the functionality to display a dialog box that allows the user to browse to and enter the name of a file to open or save
  - The **`FontDialog`** class displays a dialog box to change elements of the Font object used by your application
  - The **`PageSetupDialog`**, **`PrintPreviewDialog`**, and **`PrintDialog`** classes display dialog boxes that allow the user to control aspects of printing documents
- 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

Components

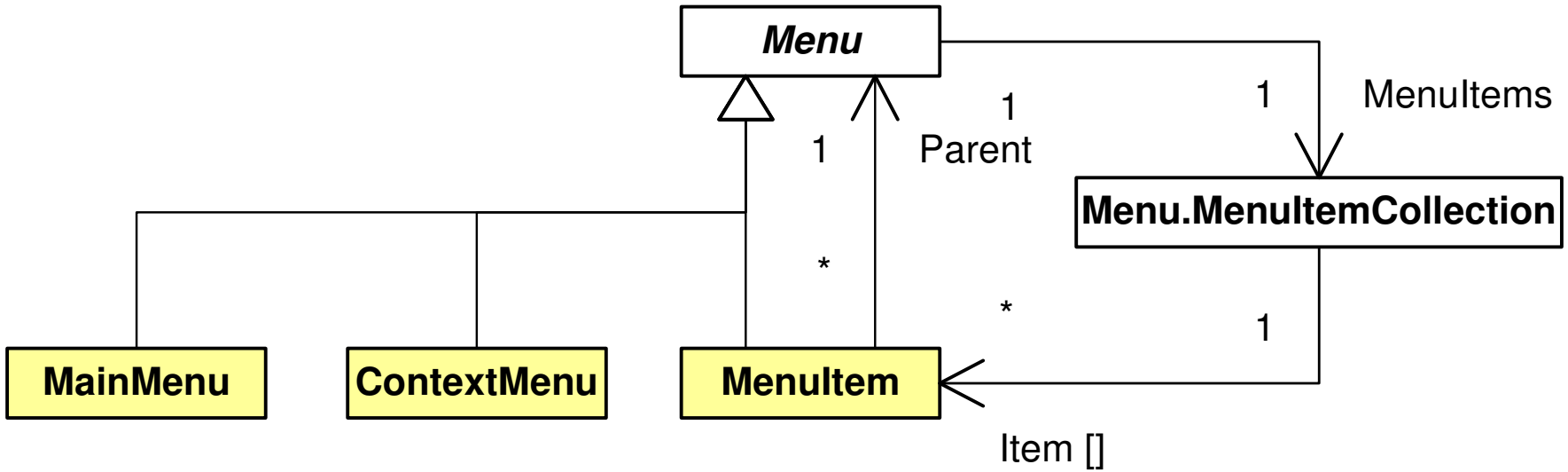
- In programming, the term **component** is generally used for an object that is reusable and can interact with other objects
- A .NET Framework `Component` satisfies those general requirements and additionally provides features such as
  - **Control over unmanaged resources**
  - **Design-time support**
    - A component can be used in a rapid application development (RAD) environment
    - A component can be added to the toolbox of Visual Studio .NET, can be dragged and dropped onto a form, and can be manipulated on a design surface
    - Note that 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

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 **`Menu`**, **`MenuItem`**, and **`ContextMenu`** classes provide the ability display menus to the user to invoke commands within an application
- The **`Help`** and **`HelpProvider`** classes enable you to display help information to the user of your applications

Working with Menu's

- **`MainMenu`**, **`ContextMenu`**, and **`MenuItem`** are derived from **`Menu`**
- **`Menu`** includes a collection of **`MenuItem`**'s



Working with Menu's

- Create a **`MainMenu`** (or **`ContextMenu`**)

```
MainMenu mainMenu = new MainMenu();
```

- Add **`MenuItem`**s to the **`MainMenu`**

```
MenuItem menuItem1 = new MenuItem("&File");
mainMenu.MenuItems.Add(menuItem1);
```

- Add sub-**`MenuItem`**s

```
MenuItem menuItem2 = new MenuItem("E&xit");
menuItem1.MenuItems.Add(menuItem2);
```

- Set **`Form`**'s **`Menu`** property to the instance of the **`MainMenu`**

```
myForm.Menu = mainMenu;
```

Working with Menu's

- Define event handlers

```
private void ExitHandler(object sender, EventArgs e)
{
  Close();
}
```

- Register event handlers

```
menuItem2.Click += new EventHandler(ExitHandler);
```