# Chapter 1 - Event-Driven Fundamentals

Ref. - Murach’s C# (7th Ed.), Ch. 1–3; Microsoft Learn: "Create a Windows Forms App with C#“; csharpkey.com (GUI tutorials)

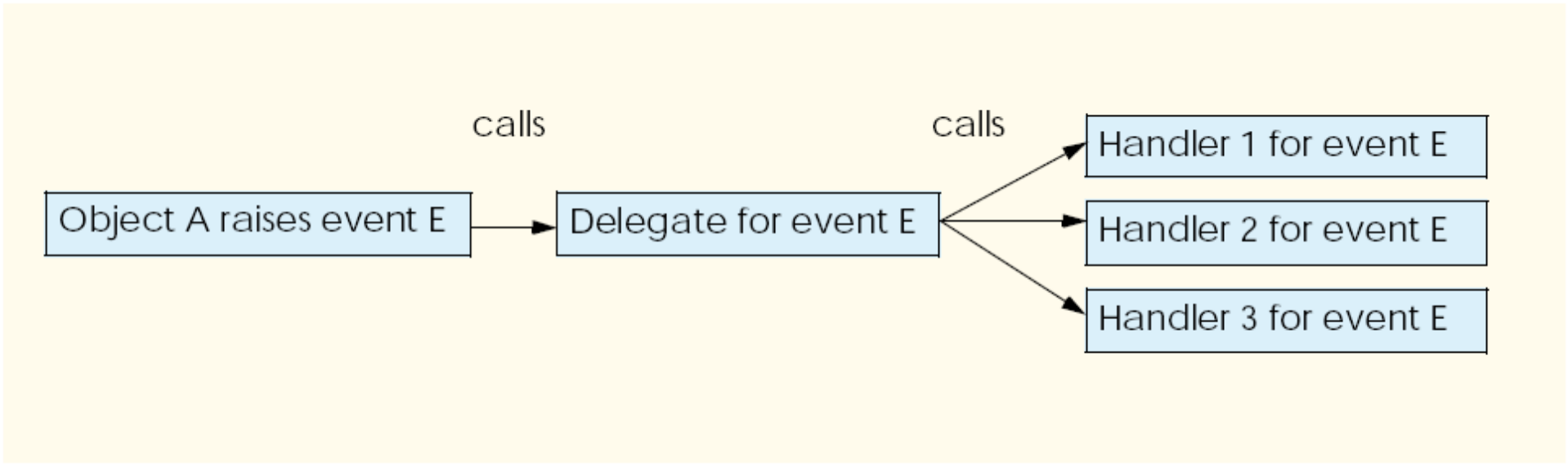
# Chapter Overview

- This chapter covers the fundamentals of event-driven programming using C# and the .NET framework.
- Focus on understanding concepts, setting up visual studio environment, and building basic applications.
- Practice in Visual Studio during labs.

# 1. Introduction to Event-Driven Programming

- **Definition**: Event-driven programming is a paradigm where the flow of a program is determined by *events* (user actions such as mouse clicks, key presses, or system-generated messages).
- Unlike procedural programming (which follows a sequential path), event-driven programs respond to "events" triggered by the user or environment.

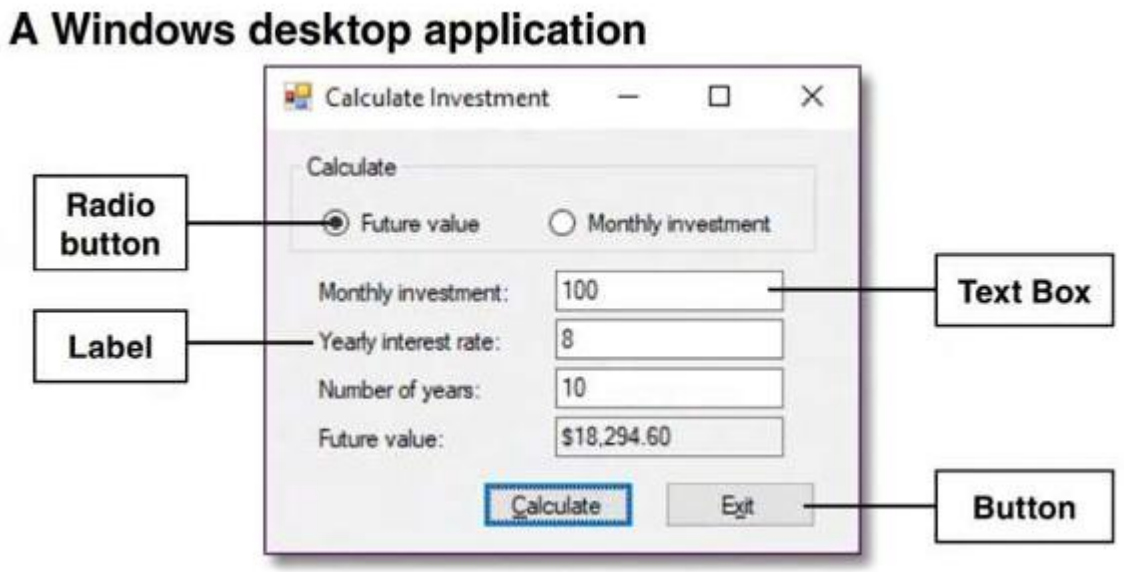
- **Key Concepts**:
  - **Event** → A signal that something has happened (e.g., button clicked).
  - **Event Handler** → A method that executes in response to an event.
  - **Delegate** → A type-safe reference that links an event to its handler.
- GUI application heavily rely on this model.



## .NET Applications Overview:

- C# use to create range of application that run on the .NET platform.
- GUI applications (Windows Forms, WPF, Web apps).
  - Forms (or WinForms) platform became popular because it provides a drag-and-drop designer that makes it easy to develop the user interface for an application
- **Windows Desktop Applications**: In a WinForms application, each Windows form (or just form) provides a user interface that lets the user interact with the application. The application consists of a single form, or many applications require more than one form.

- As part of the user interface, a Windows Forms application uses controls like the ones shown in the figure (e.g., *radio buttons, labels, text boxes, buttons*).
- These controls allow the user to interact with the application.
- Unlike procedural programming (step-by-step flow), event-driven programs wait for user interaction.



# Platforms for Developing Windows Desktop Applications:

- **Windows Forms (WinForms)**: Provides a drag-and-drop designer that automatically generates the code that defines the user interface (*Supported by Windows 7 and later*).
- **Windows Presentation Foundation (WPF)**: Requires the programmer to write XAML (Extensible Application Markup Language) that defines the user interface, but provides an enhanced user experience (*Supported by Windows 7 and later*).
- **Universal Windows Platform (UWP)**: Requires the programmer to write XAML that defines the user interface, but provides a way to create responsive applications that support various screen sizes. (*Supported by any Windows 10 device*).

## Three platforms for developing Windows desktop applications with .NET

| Type | Description |
|---|---|
| Windows Forms (WinForms) | Provides a drag-and-drop designer that automatically generates the code that defines the user interface. Supported by Windows 7 and later. |
| Windows Presentation Framework (WPF) | Requires the programmer to write XAML (Extensible Application Markup Language) that defines the user interface, but provides an enhanced user experience. Supported by Windows 7 and later. |
| Universal Windows Platform (UWP) | Requires the programmer to write XAML that defines the user interface, but provides a way to create responsive applications that support various screen sizes. Supported by any Windows 10 device. |

## Two platforms for developing web applications with .NET

| Type | Description |
|---|---|
| ASP.NET Web Forms | Provides a drag-and-drop designer that automatically generates the code for the web forms that define the pages of the web application. |
| ASP.NET Core MVC | Requires the user to write code that uses the Model-View-Controller (MVC) design pattern to define the pages of the web application. |

# The .NET Framework and .NET Core

## 1) Overview

- **.NET Framework** and **.NET Core** are both platforms for developing and running applications.
- They work in very similar ways.

## 2) Key Differences

- **.NET Framework**
  - Older platform.
  - Supports **Windows only**.
  - Development stops at **version 4.8**.
- **.NET Core**
  - Newer platform.
  - **Cross-platform** → runs on Windows, macOS, and Linux.
  - Represents the **future of .NET development**.
  - The next version after .NET Core 3.1 is **.NET 5** (dropping "Core" in the name).
  - .NET 5 combines the best of .NET Framework and .NET Core, offers migration support, and lays the foundation for future growth.
- **Class Libraries**
  - Both platforms provide **class libraries** — collections of prewritten code (classes).
  - Examples:
    - **Windows Forms classes** → build Windows Forms apps.
    - **ASP.NET / ASP.NET Core classes** → build web apps.
    - **EF / EF Core classes** → access databases.
    - Other classes → manage security, file access, and many other tasks.

## 4) Common Infrastructure

- Both platforms provide the same basic **infrastructure** so that applications written in a .NET language (like C#) can run.
- This includes:
  - **Common Language Runtime (CLR)** → executes .NET applications.
  - **.NET Languages** → e.g., C#, VB.NET, F#.
  - **Compilers** → turn source code into intermediate form.

## 5) Common Language Runtime (CLR)

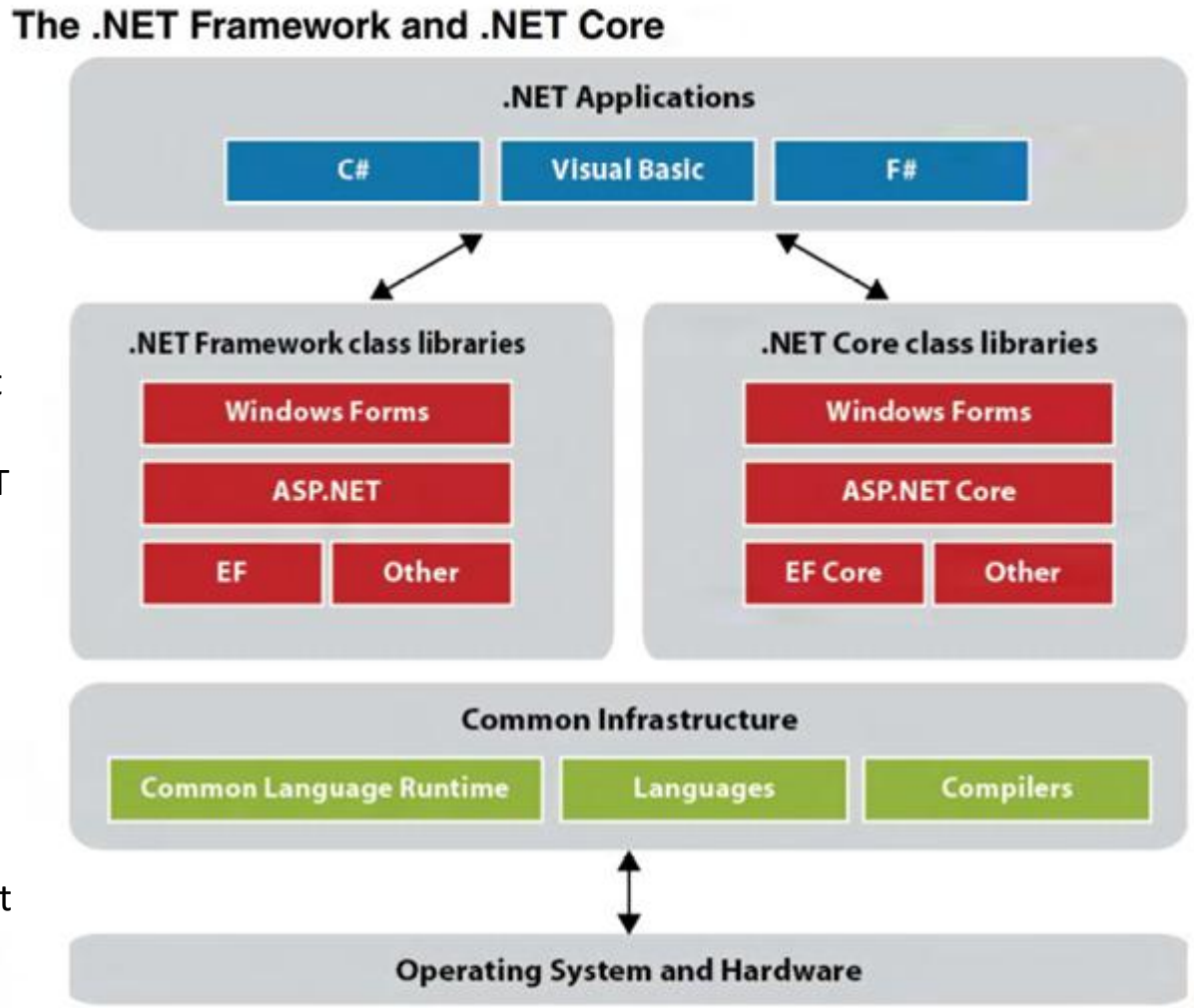
- The CLR provides all services needed to execute .NET applications.
- It works:
  - All .NET languages compile to a **common intermediate language (IL)**.
  - CLR executes IL on the target platform.
- **Common Type System (CTS)**: CLR also provides CTS
  - Defines shared data types for all .NET languages.
- Because all of the .Net applications are managed by the CLR, they are sometimes referred to as **managed applications**.

# More on the .NET Framework and .NET Core

- In the same way that you saw **Windows Forms classes**, **ASP.NET classes**, and **Entity Framework (EF) classes**, **ADO.NET classes** are also part of the .NET class libraries.
- They provide the **fundamental data access layer**: connections, commands, data readers, data adapters, and datasets.
- **In .NET Framework**
  - ADO.NET is the standard way to connect to SQL Server and other databases.
  - You can use namespaces like System.Data, System.Data.SqlClient.
- **In .NET Core / .NET 5+**
  - ADO.NET is still supported.
  - The System.Data.Common base classes and System.Data.SqlClient provider are available.
  - In fact, **Entity Framework Core** is built on top of ADO.NET — it uses ADO.NET under the hood to execute SQL commands.
- **Note**
  - For **low-level, high-performance, or legacy projects**, ADO.NET is often preferred.
  - For **modern apps**, EF Core is usually the first choice because it provides object-relational mapping (ORM), but ADO.NET remains a solid option for direct SQL control.

# Description

- Windows Forms applications do not access the operating system or computer hardware directly. Instead, they use services of the .NET Framework or .NET Core, which in turn access the operating system and hardware.
- The .NET Framework and .NET Core provide class libraries that contain pre-written code know as classes that are available to all of the NET programming languages. These class libraries consist of thousands of classes, but you can create simple .NET applications once you learn how to use just a few of them.
- The .NET Framework and .NET Core use a common infrastructure that includes the Common Language Runtime (CLR), the languages, and the compilers.
- The CLR manages the execution of .NET programs by coordinating essential functions such as memory management, code execution, security, and other services. Because .NET applications are managed by the CLR, they are called managed applications.
- The Common Type System(CTS) is a component of the CLR that ensures that all .NET applications use the same basic data types no matter what programming languages are used to develop the applications.



# 2. Visual Studio 2019/2022 Setup

- Visual Studio is an Integrated Development Environment (IDE) that supports programming languages and application development platforms.
- As of September 2025, the latest stable version is Visual Studio 2022.

| Edition | Description |
| --- | --- |
| Community | A free edition that's appropriate for hobbyists, students, and individual developers building non-enterprise applications. |
| Professional | Designed for small teams of professionals building non-enterprise applications. Includes tools for analyzing code and collaborating with teams. |
| Enterprise | Designed for teams of any size who want to build complex, scalable enterprise applications. |

# Installation Steps

- **Download**: Go to visualstudio.microsoft.com/downloads. Select Visual Studio 2022 Community (free) or other editions.
- **System Requirements**: Windows 10/11 (64-bit recommended), 8 GB RAM minimum, 210 GB disk space for full install. Check full requirements on the site.
- **Workloads**: During installation, select:
  - **.NET desktop development** (for Windows Forms and C# GUI apps).
  - Optionally, **ASP.NET and web development** for web apps.
  - This installs the .NET Framework and .NET (formerly .NET Core) for cross-platform support.
- **Additional Options**: Choose to install SQL Server Express LocalDB for database support if needed.
- **Run Installer**: Follow the on-screen prompts. Installation may take 30-60 minutes.
- **Post-Install**: Launch Visual Studio, sign in with a Microsoft account (optional for Community), and update if prompted.

**Notes on Visual Studio installation**:

- When you install Visual Studio, you can choose to install the .NET Framework or .NET Core.
- These platforms provide a way to run applications written in other .NET languages such as Visual Basic or C#.
- Visual Studio for Mac runs on macOS but doesn't provide a way to develop desktop applications for Windows.
- .NET Framework is for Windows-only apps; .NET (unified since .NET 5+) supports cross-platform (Windows, macOS, Linux).
- Troubleshooting: If issues arise, refer to learn.microsoft.com/visualstudio/releases/2022/release-notes.

# 3. Building a Simple GUI App (e.g., "Hello World" with Button Events)

(*Reading practice - Chapter 1 - VS setup-Build first app-explore vs environment*)

- A basic Windows Forms app to demonstrate event-driven concepts.

**Steps:**

1. **Create a Project**: In Visual Studio, go to File > New > Project.

Select "Windows Forms App (.NET Framework)" or ".NET" for cross-platform. Name it "HelloWorld".

**2. Design the Form**: Use the drag-and-drop designer (Form Designer) to add controls.

- Drag a **Label** and set its Text to "Enter your name:".
- Drag a **TextBox** for user input.
- Drag a **Button** and set its Text to "Say Hello".

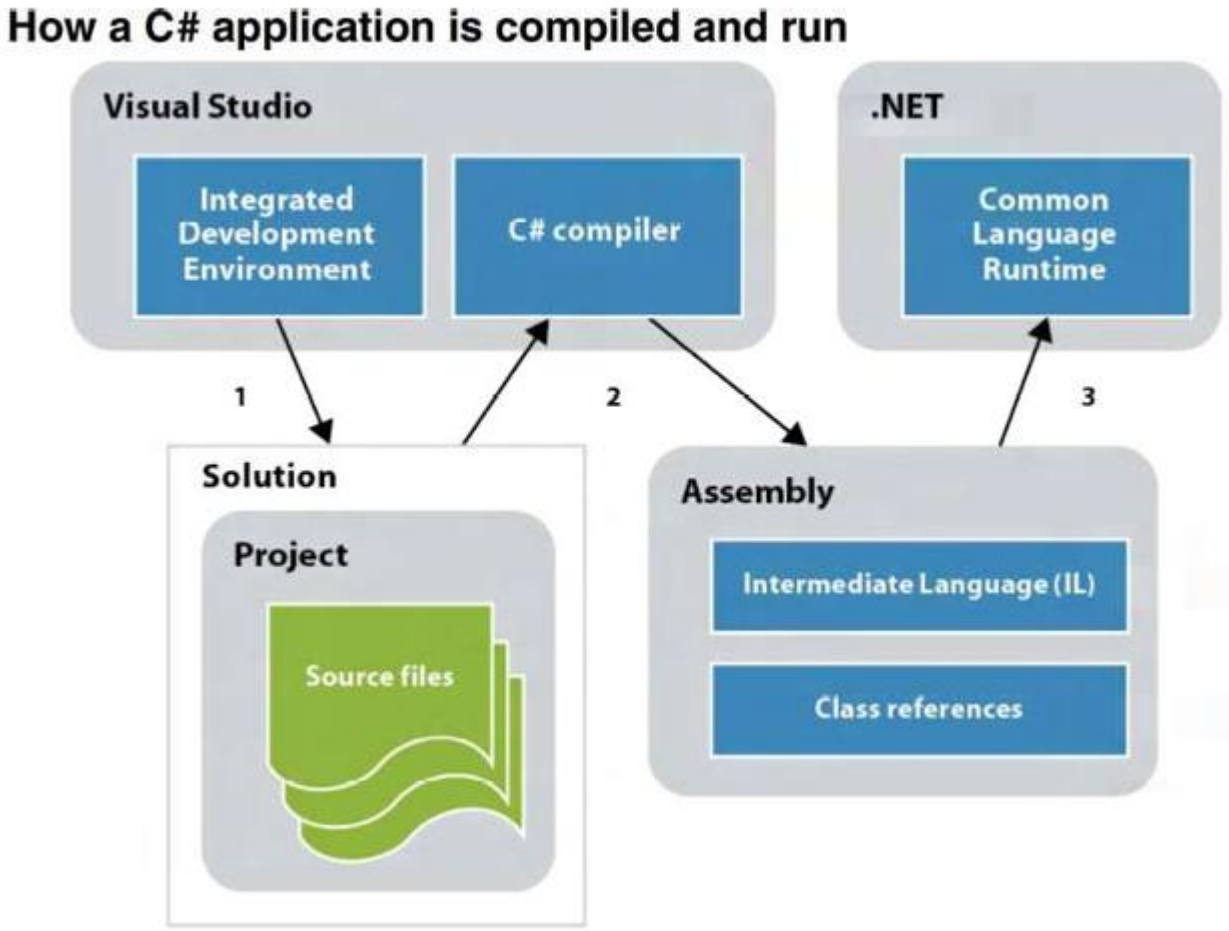
**3. Handle Events**: Double-click the Button to create a Click event handler. Add C# code:

```
private void button1_Click(object sender, EventArgs e)
{
    label1.Text = "Hello, " + textBox1.Text + "!";
}
```

**4. Build and Run**: Press F5 or Debug > Start Debugging. The app runs, and clicking the button triggers the event to update the label.

# Compilation Process

- Programmer uses Visual Studio to create a project with C# source files.
- C# compiler compiles the source into Microsoft Intermediate Language (MSIL) or Intermediate Language (IL).
- Assembly (executable .exe or .dll) is run by .NET's Common Language Runtime (CLR), converting IL to native code.



# 4. Intrinsic Controls (Buttons, Labels, Textboxes)

- Intrinsic controls are built-in UI elements in Windows Forms for creating GUIs.
- **Intrinsic controls** because they form the **core elements** of most GUI applications.
  - **Buttons**: Trigger actions on click (e.g., submit data).
  - **Labels**: Display static or dynamic text (e.g., prompts or results).
  - **Textboxes**: Allow user input (e.g., numbers or text).
- Controls are dragged from the Toolbox to the form.
- They respond to events like Click (for Button) or TextChanged (for TextBox).

| Control | Purpose |
|---|---|
| Button | Executes code on click (event-driven action). |
| Label | Displays read-only text. |
| TextBox | Accepts user input; can trigger events on change. |

- A **Button**
  It typically appears as a clickable rectangle with a caption (text).

## *Key Properties*

- Text → the caption displayed on the button.
- Enabled → determines whether the button can be clicked.
- Visible → shows or hides the button.
- BackColor / ForeColor → button colors.
- Font → typeface, size, and style of the caption.

## *Key Events*

- Click → triggered when the user clicks the button.
- MouseEnter / MouseLeave → when the pointer enters/leaves the button area.
- KeyPress → when a key is pressed while the button has focus

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

A **Label**
It cannot accept input — it's a passive, non-editable control.

### *Typical Uses*

- Displaying instructions: *"Enter your name:"*
- Showing status updates: *"Processing…"*
- Describing input fields next to textboxes.

### *Key Properties*

- Text → the text displayed.
- AutoSize → automatically resizes the label to fit its text.
- BorderStyle → can give the label a border for emphasis.
- TextAlign → aligns text (left, center, right)

```
lblStatus.Text = "Ready to process.";
```

A **TextBox** - a single line (or multiple lines) of text.

*Key Properties*

- Text → the text contained in the box.
- Multiline → true allows multi-line input.
- PasswordChar → masks characters (e.g., for password input).
- ReadOnly → prevents editing while still displaying text.
- MaxLength → sets the maximum number of characters allowed.

*Key Events*

- TextChanged → triggered whenever text inside changes.
- KeyPress → triggered for each key typed.
- Leave → triggered when the user leaves the textbox.

```
private void txtName_TextChanged(object sender, EventArgs e)
{
    lblStatus.Text = "You typed: " + txtName.Text;
}
```

**Best Practice**

1. **Naming Convention**: Use meaningful names.
   btnSubmit, lblStatus, txtName (instead of generic names like button1).
2. **Validation**: Always validate TextBox input before using it (e.g., check empty input, numeric validation).
3. **Accessibility**: Use Labels properly to guide users.
4. **Consistency**: Keep button placement consistent (e.g., OK on the left, Cancel on the right).

# 5. Properties, Methods, and Events

- **Properties**: Attributes of controls that define **appearance/behavior** (e.g., Text, Size, Location). Set in the Properties window.
- **Methods**: Actions performed by controls or code (e.g., MessageBox.Show() to display a dialog).
- **Events**: Triggers like Button.Click or TextBox.TextChanged. Handle them by writing code in event handlers.
- In the Visual Studio IDE, the Properties window lets you set control properties (e.g., Text = "Calculate").
- Events are wired in the code editor: Double-click a control to generate an event handler method.
  - Example: Button Click event calls a method to update a Label's Text property.
- **Lab Practice**:
  - In your "Hello World" app, experiment with properties (change Button color) and add a KeyPress event to the TextBox.
  - In your "Hello World" app, modify it to handle a second event (e.g., clear button).

**General Notes on Control Properties in C# WinForms**

- **Properties** in WinForms controls are **public members** that wrap around private fields, often with validation logic.
- They control **appearance**, **behavior**, and **interaction** of a control.
- Many properties are **inherited** from the base Control class (System.Windows.Forms.Control), while others are **specific** to a particular control.

•**Property Categories**:

- **Visual properties**: Font, Color, Size, Location.
- **Behavioral properties**: Enabled, TabStop, ReadOnly.
- **Accessibility properties**: AccessibleName, AccessibleDescription.
- **Data-related properties**: Text, Tag, DataBindings.

**Advanced Properties of the Button Control**

| Property | Description & Advanced Use |
|---|---|
| DialogResult | Useful in modal forms (e.g., OK, Cancel). When clicked, closes the form and sets the form's DialogResult. |
| FlatStyle | Controls rendering style (Standard, Flat, Popup, System). Allows for custom look. |
| Image & ImageAlign | Attach an icon or image to the button, and align it relative to text. Common in toolbars. |
| TextImageRelation | Defines how text and image are placed (Overlay, ImageBeforeText, etc.). |
| UseMnemonic | Allows keyboard shortcut with & (e.g., &Save → Alt+S triggers button). |
| Anchor & Dock | Define how the button resizes or sticks when the form is resized. |
| TabIndex | Sets keyboard navigation order. |

# End of Chapter One