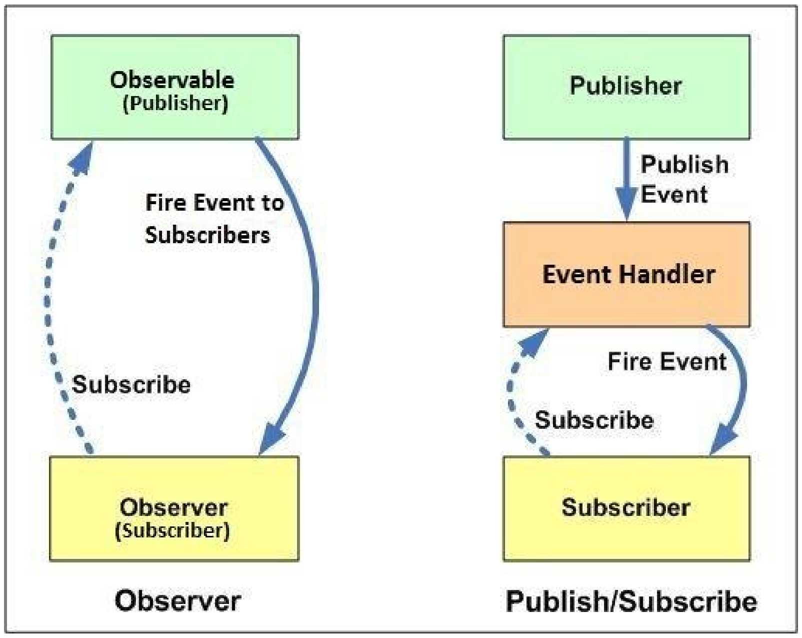
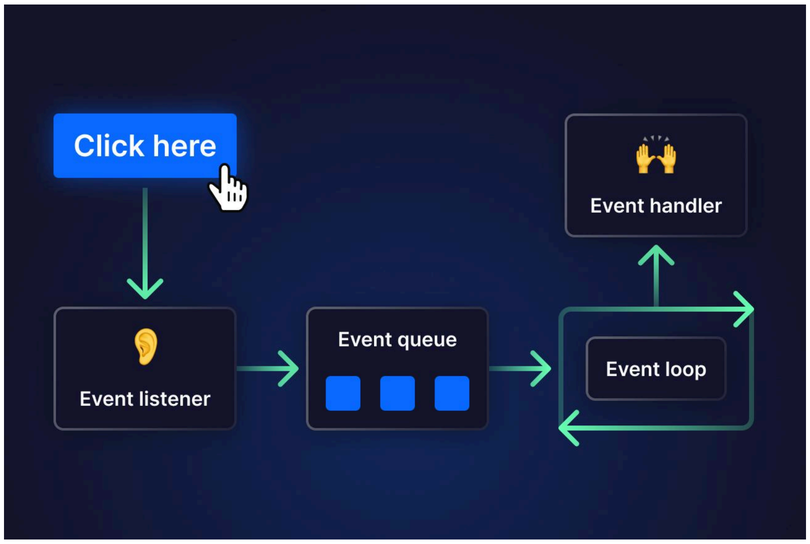
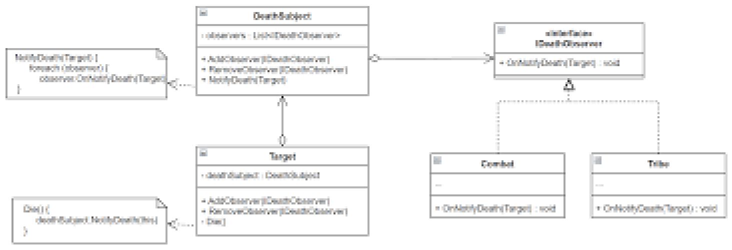
📘 Events in C#

1. Definition of Events






DeathSubject
- observers : List<IDeathObserver>
+ AddObserver(IDeathObserver)
+ RemoveObserver(IDeathObserver)
+ NotifyDeath(Target)
NotifyDeath(Target) { foreach (observer) { observer.OnNotifyDeath(Target) } }
«interface» IDeathObserver
+ OnNotifyDeath(Target) : void
Target
- deathSubject : DeathSubject
+ AddObserver(IDeathObserver)
+ RemoveObserver(IDeathObserver)
+ Die()
Die() { deathSubject.NotifyDeath(this) }
Combat
...
+ OnNotifyDeath(Target) : void
Tribe
...
+ OnNotifyDeath(Target) : void

An **event** in C# is a language feature that allows a class to **notify other classes** when something significant happens.

Events follow the **Publisher–Subscriber model**:

- The **publisher** defines and raises the event
- The **subscriber** listens and reacts to the event

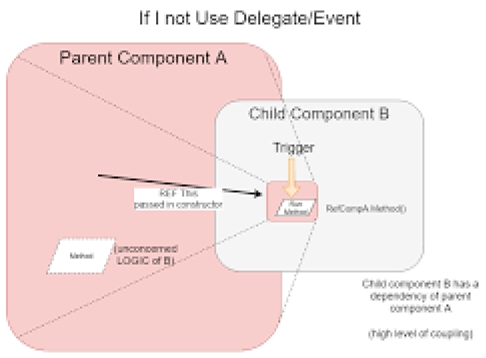
Events are commonly used in:

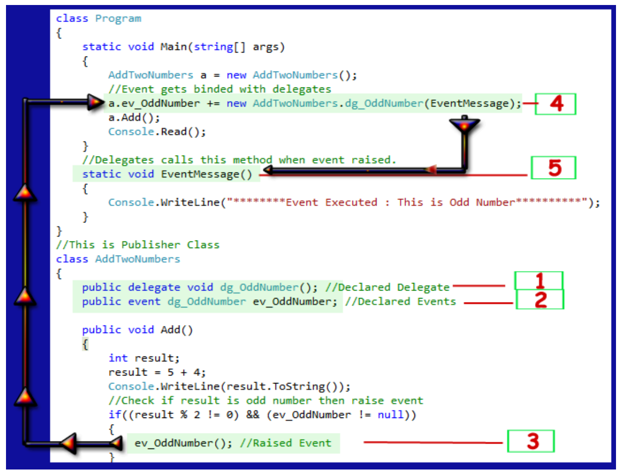
- GUI applications (button clicks)
- Real-time notifications
- Status monitoring
- Logging systems
- Asynchronous workflows

Key Points

- Events represent **actions or occurrences**
- They enable **loose coupling** between components
- Multiple subscribers can listen to a single event

2. The `event` Keyword





```
class Program
{
    static void Main(string[] args)
    {
        AddTwoNumbers a = new AddTwoNumbers();
        //Event gets binded with delegates
        a.ev_OddNumber += new AddTwoNumbers.dg_OddNumber(EventMessage);
        a.Add();
        Console.Read();
    }
    //Delegates calls this method when event raised.
    static void EventMessage()
    {
        Console.WriteLine("********Event Executed : This is Odd Number**********");
    }
}
//This is Publisher Class
class AddTwoNumbers
{
    public delegate void dg_OddNumber(); //Declared Delegate
    public event dg_OddNumber ev_OddNumber; //Declared Events

    public void Add()
    {
        int result;
        result = 5 + 4;
        Console.WriteLine(result.ToString());
        //Check if result is odd number then raise event
        if((result % 2 != 0) && (ev_OddNumber != null))
        {
            ev_OddNumber(); //Raised Event
        }
```

The `event` keyword in C# is used to **declare an event** and control how it can be accessed.

Why the `event` keyword is important

- Prevents external classes from invoking the event directly
- Ensures that only the declaring class can raise the event
- Allows subscription (`+=`) and unsubscription (`-=`) only

Syntax

```
public event EventHandler MyEvent;
```

Without `event`

External classes could invoke the delegate directly (unsafe).

With `event`

Only subscription and unsubscription are allowed (safe and controlled).

🔧 IMPLEMENTATION SECTION

(For remaining topics as requested)

3. Declaring and Raising Events (Implementation)

Declaring an Event

```
public event EventHandler DataProcessed;
```

Raising an Event

```
DataProcessed?.Invoke(this, EventArgs.Empty);
```

✔ `?.Invoke()` prevents `NullReferenceException`
✔ Event is raised only inside the declaring class

4. How to Declare an Event (Implementation)

Step 1: Use Built-in Delegate

```
public event EventHandler ProcessCompleted;
```

Step 2: (Optional) Custom EventArgs

```
public class ProcessEventArgs : EventArgs
{
    public string Message { get; set; }
}

public event EventHandler<ProcessEventArgs> ProcessCompleted;
```

5. How to Raise an Event (Implementation)

Best practice is to raise events inside a protected virtual method:

```
protected virtual void OnProcessCompleted(string message)
{
    ProcessCompleted?.Invoke(this, new ProcessEventArgs { Message =
message });
}
```

6. Subscribing to and Handling Events (Implementation)

Subscribing

```
publisher.ProcessCompleted += OnProcessCompleted;
```

Event Handler

```
void OnProcessCompleted(object sender, ProcessEventArgs e)
{
    Console.WriteLine(e.Message);
}
```

Unsubscribing

```
publisher.ProcessCompleted -= OnProcessCompleted;
```

7. One Full-Blown Example (Covers ALL Remaining Topics)

Scenario

A **FileDownloader** raises an event when the download completes.
A **Logger** subscribes and reacts.

🔹 Complete Working C# Program

```
using System;

namespace EventsDemo
{
    // Custom EventArgs
    public class DownloadEventArgs : EventArgs
    {
        public string FileName { get; set; }
    }

    // PUBLISHER
    public class FileDownloader
    {
        // Declare event
        public event EventHandler<DownloadEventArgs>
DownloadCompleted;

        public void StartDownload(string fileName)
        {
            Console.WriteLine($"Downloading {fileName}...");
            OnDownloadCompleted(fileName);
        }

        // Raise event
        protected virtual void OnDownloadCompleted(string fileName)
        {
            DownloadCompleted?.Invoke(this, new DownloadEventArgs
            {
                FileName = fileName
            });
        }
    }

    // SUBSCRIBER
    public class Logger
    {
        public void OnDownloadCompleted(object sender,
DownloadEventArgs e)
        {
            Console.WriteLine($"Download completed: {e.FileName}");
        }
    }

    class Program
    {
        static void Main()
        {
            FileDownloader downloader = new FileDownloader();
            Logger logger = new Logger();

            // Subscribe
            downloader.DownloadCompleted +=
logger.OnDownloadCompleted;

            downloader.StartDownload("report.pdf");

            // Unsubscribe
            downloader.DownloadCompleted -=
logger.OnDownloadCompleted;

            Console.ReadLine();
        }
    }
}
```

🧠 Output

```
Downloading report.pdf...
Download completed: report.pdf
```

📌 Summary

Topic	Covered
Definition of events	✅ Documentation + Images
event keyword	✅ Documentation + Images
Declaring events	✅ Implementation
Raising events	✅ Implementation
Subscribing & handling	✅ Implementation
Full example	✅ Yes