C# EVENTS

WHAT IS EVENT

- C# events are mechanisms that allow objects to notify about events that have occurred within them. Events can be user actions (e.g., button press) or system events (e.g., file loading completion) and they enable other objects to react to these events.

EVENT DECLARATION AND TRIGGERING

- An object that will trigger an event declares the event using the event keyword and points to a method that will be invoked when the event occurs.

```
public class Button
{
    public event EventHandler Click; // Declare an event

    public void OnClick()
    {
        Click?.Invoke(this, EventArgs.Empty); // Trigger the event
    }
}
```

EVENT HANDLING (EVENT HANDLERS)

- Users register as listeners to events by using an event handling method (event handler). It's a regular method signature that matches EventHandler or a custom delegate type. The event handler method is invoked when the event happens

```
public void Button_Click(object sender, EventArgs e)
{
    // Event handling logic here
}

// Register an event handler
button.Click += Button_Click;
```

EVENT ATTACHMENT AND DETACHMENT

- To attach or detach an event handler, you use the += and -= operators:

```
button.Click += Button_Click;    // Attach an event handler
button.Click -= Button_Click;    // Detach an event handler
```

C# DELEGATES

- C# delegates are references to methods (or multiple methods), allowing you to create flexible and dynamic invocations to functions. Delegates are value types that allow passing functions as parameters and storing them as variables.

DELEGATE DECLARATION AND USAGE.

- Define a delegate using the delegate keyword and specify the signature of the method it can invoke. Delegates can be invoked using the Invoke method or simply by calling them as functions.

```
public delegate void MyDelegate(string message);

// Define a method that can be invoked by the delegate
static void PrintMessage(string message)
{
    Console.WriteLine(message);
}

MyDelegate myDelegate = PrintMessage; // Assign a function to the delegate
myDelegate("Hello, delegates!"); // Invoke the delegate
```

BUILT-IN DELEGATES

- C# provides some built-in delegates like «Action» and «Func» that allow you to specify functions without defining a custom delegate type

```
Action<string> action = PrintMessage;
Func<int, int, int> add = (x, y) => x + y;
```

```
void ProcessData(List<int> data, Action<int> processor)
{
    foreach (var item in data)
    {
        processor(item);
    }
}

ProcessData(dataList, PrintMessage);
ProcessData(dataList, item => Console.WriteLine(item * 2));
```

C# THREADING

- Threading in C# is the process of executing multiple actions concurrently or nearly concurrently by dividing the program's work into different threads. Each thread is an execution path capable of executing a specific section of code

WHAT IS THREADING.

- Threading allows programs to use available processor resources effectively by executing multiple tasks concurrently. For example, a program might have one thread responsible for the user interface and another thread for working with a database.

CREATING AND MANAGING THREADS

- Threads can be created using the «Thread» class from the «System.Threading» namespace

```
Thread thread = new Thread(DoWork);
thread.Start(); // Start thread execution
```

THREAD SYNCHRONIZATION

- Threads can contend for shared resources, leading to issues like incorrect data access or problems. To avoid such situations, synchronization is used, often using constructs like the lock statement

PARALLEL THREDING

- Parallel threading is a technique that allows tasks to be executed concurrently using all available processor cores. This can be achieved using the «Parallel» class from the «System.Threading.Tasks» namespace, using functions like «Parallel.For» and «Parallel.ForEach»

```
Parallel.For(fromInclusive: 0, toExclusive: 10, body: i :int =>
{
    Console.WriteLine($"Task {i} is running on thread {Thread.CurrentThread.ManagedThreadId}");
});

List<int> dataList = GetLargeListOfData();
Parallel.ForEach(dataList, body: item :int =>
{
    ProcessData(item);
});
```