Delegados, eventos y lambdas en C#

Delegados

Un delegado es un tipo seguro que apunta a métodos que tienen un prototipo firma concreto. En otras palabras, es un puntero a método, seguro y tipado en C#.

Analogía: un delegado es como un mando a distancia que puede apuntar a distintos métodos. Cuando aprietas el botón, llama al método conectado.

Sintaxis básica

```
class Amable{
    public void Saluda(String nombre) {
        Console.WriteLine($"Hola {nombre}");
    }
    public void Despide(String nombre) {
        Console.WriteLine($"Adiós {nombre}");
    }
}

static class Program {
    delegate void MiDelegado(String s);

    static void Main(){
        Amable amable = new Amable();

        MiDelegado seAmableCon = amable.Saluda;
        seAmableCon("Sisebuto"); // Hola Sisebuto

        seAmableCon += amable.Despide;
        seAmableCon("Sisebuto"); // Hola Sisebuto
                                 // Adiós Sisebuto

        seAmableCon -= amable.Saluda;
        seAmableCon("Alarico");  // Adiós Alarico
    }
}
```

La declaración del delegado es similar a la del prototipo del método al que puede apuntar:

```
    public void Saluda(String nombre) {
        Console.WriteLine($"Hola {nombre}");
    }
…
    delegate void MiDelegado(String s);
```

Una vez asignado el delegado, su invocación es análoga a la de un método:

```
        MiDelegado seAmableCon = amable.Saluda;
        seAmableCon("Sisebuto");
```

Multicast: un delegado puede apuntar y desapuntar a varios métodos a la vez:

```
        seAmableCon += amable.Despide;
        seAmableCon("Sisebuto");  // Hola Sisebuto
                                  // Adiós Sisebuto

        seAmableCon -= amable.Saluda;
        seAmableCon("Alarico");   // Adiós Alarico
```

Delegados genéricos predefinidos

C# cuenta con varios delegados genéricos ya definidos que en la práctica hacen innecesario que tengamos que definir nuestros propios delegados.

```
class Persona {
    private int _edad;

    public Persona(int edad) { _edad = edad; }

    public int Edad() { return _edad; }
    public bool MayorQue(int edad) { return _edad > edad; }
    public void Saluda(String nombre)
      { Console.WriteLine($"Hola {nombre}"); }
}

static class Program {
    static void Main(){
        Persona persona = new Persona(23);

        Func<int> tuEdad = persona.Edad;
        Predicate<int> esMayorQue = persona.MayorQue;
        Action<String> unSaludo = persona.Saluda;

        Console.WriteLine($"Edad: {tuEdad()}"); // 23
        Console.WriteLine($"Mayor de edad: {esMayorQue(17)}"); // True
        Console.Write("Saluda a Alarico: ");
        unSaludo("Alarico"); // Hola Alarico
    }
}
```

- *Action<T…>* es un delegado sobre un método que no devuelve valor (void) y puede tener 0 o más parámetros.
- *Func<T..., TResult>* es un delegado a un método que devuelve un valor y puede tener 0 o más parámetros.
- *Predicate<T>* es un delegado a un método que recibe un único parámetro y devuelve bool.

```
class Program {
    static void MostrarProducto(int x, int y)
        { Console.WriteLine($"El producto es: {x * y}");}

    static int Sumar(int a, int b)
        { return a + b; }

    static bool Positivo(int a)
        { return a > 0; }

    static void Main() {
        Action<int, int> accionProducto = MostrarProducto;
        accionProducto(3, 5);

        Func<int, int, int> funcionSuma = Sumar;
        int resultado = funcionSuma(4, 7);

        Predicate<int> predicadoPositivo = Positivo;
        bool esPositivo = predicadoPositivo(3);
    }
}
```

Eventos

Un evento es un mecanismo que permite que un objeto envíe una notificación a otros objetos cuando algo (*evento*) ocurre.
Internamente usa un delegado, pero solo la clase propietaria puede dispararlo.

```
public class Publicador {
    public event Action<String> EventoNotificacion;

    public void Disparar()
        { EventoNotificacion?.Invoke("¡Evento disparado!"); }
}

public class Suscriptor {
    public void ResponderEvento(string mensaje)
        { Console.WriteLine("Recibido: " + mensaje); }
}

class Programa {
    static void Main() {
        var publicador = new Publicador();
        var suscriptor = new Suscriptor();

        publicador.EventoNotificacion += suscriptor.ResponderEvento;
        publicador.Disparar(); // Recibido: ¡Evento disparado!
    }
}
```

Lambdas

Las lambdas son simplemente una forma concisa y directa de escribir funciones anónimas, es decir, funciones que no tienen nombre y que se definen justo donde se necesitan.

Sintaxis básica

(parametros) => expresion

- *Parámetros*: entre paréntesis y separados por comas.
- =>: operador lambda.
- *Expresión* o *bloque*: lo que la función debe ejecutar o devolver.

```
Func<int, int> cuadrado = x => x * x;
Console.WriteLine(cuadrado(5)); // 25

Func<int, int, int> sumar = (a, b) => a + b;
Console.WriteLine(sumar(3, 4)); // 7

Func<int, int, int> multiplicar = (x, y) => {
    int producto = x * y;
    Console.WriteLine($"Multiplicando: {x} por {y}");
    return producto;
};
int producto = multiplicar(3, 4); // 12
```

Utilidad

- Ahorra código: no se necesita escribir todo un método. Sólo para algo pequeño.
- Legibilidad: el código queda más claro y directo.
- Uso instantáneo: se usan para pasar funciones como parámetro (por ejemplo, al filtrar, mapear o reducir colecciones en LINQ).