

¿Qué es un delegado?

- •En las comunicaciones de eventos, el remitente del evento **no sabe** qué objeto o método recibirá los eventos que provoca.
- •Se necesita un **intermediario** entre el origen y el receptor.
- NET Framework define un tipo especial
 (Delegado) que proporciona la funcionalidad de un puntero a función.
- •Un delegado es una clase que **puede guardar una referencia a un método**.



Propósitos de los delegados

Creados para las situaciones en las que se necesita llevar a cabo una acción pero no se sabe de antemano qué método llamar o sobre cuál objetivo invocarla.

Se crea un delegado y se deja que los detalles particulares sean establecidos más adelante.



Declaración de un delegado de eventos

public delegate void AlarmEventHandler(object sender, AlarmEventArgs e);

- •Por convención, los delegados de evento de .NET Framework tienen dos parámetros, el origen que provocó el evento y los datos del evento.
- •Una declaración de delegado es suficiente para definir una clase de delegado.
- •La declaración no lleva { } y se puede declarar dentro de otro tipo (clase, interfaz) o por fuera de una clase.

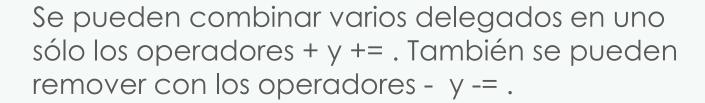


Declaración de un delegado de eventos

- •Un delegado se puede instanciar.
- •Después de ser instanciado, se le pueden añadir nuevos métodos.
- •Una instancia de un delegado puede verse como la representación de él o los métodos que contiene.
- •Internamente, una instancia de un delegado guarda una lista de invocación con todos los métodos que representa, organizados en el mismo orden en el que fueron añadidos.



Multicast



Con los operadores +, +=, - y -= también se pueden agregar y quitar métodos a los delegados.

Los métodos y delegados se ejecutan en el orden en los que fueron agregados.



Multicast

```
public delegate void MyDelegate(int i);
class TestClass {
    static void Double(int val) {
       Console.WriteLine("Ejecutando Double: " + (val * 2));
   void Triple(int val) {
       Console.WriteLine("Ejecutando Triple: " + (val * 3));
    public static void Main() {
       TestClass tc = new TestClass();
       MyDelegate d1;
       d1 = TestClass.Double;
        Console.WriteLine("-----");
       d1(3);
        d1 += tc.Triple;
        Console.WriteLine("-----");
       d1(2);
        d1 += TestClass.Double;
        Console.WriteLine("----");
       d1(4);
        Console.ReadKey();
```

Multicast

```
Ejecutando Double: 6
Ejecutando Double: 4
Ejecutando Triple: 6
Ejecutando Double: 8
Ejecutando Triple: 12
Ejecutando Double: 8
```



Combinando delegados distintos

```
delegate void MyDelegate(string s);
class MyClass{
    public static void Hello(string s){
        Console.WriteLine(" Hello, {0}!", s);
    public static void Goodbye(string s){
        Console.WriteLine(" Goodbye, {0}!", s);
    public static void Main(){
        MyDelegate a, b, c, d;
        a = new MyDelegate(Hello);
        b = new MyDelegate(Goodbye);
        c = a + b;
        d = c - a;
        Console.WriteLine("Invoking delegate a:");
        a("A");
        Console.WriteLine("Invoking delegate b:");
        b("B");
        Console.WriteLine("Invoking delegate c:");
        c("C");
        Console.WriteLine("Invoking delegate d:");
        d("D");
```



Combinando delegados distintos

```
Invoking delegate a:
Hello, A!
Invoking delegate b:
  Goodbye, B!
Invoking delegate c:
 Hello, C!
  Goodbye, C!
Invoking delegate d:
  Goodbye, D!
```



Delegado Action<T>

- •Este delegado encapsula un método que tiene un solo parámetro y no devuelve un valor.
- •Se puede usar el delegado Action<T> para pasar un método como parámetro sin declarar explícitamente un delegado personalizado. El método encapsulado debe corresponder a la firma del método definida por el delegado.



Sintaxis de Action

```
public delegate void Action<in T>(
          T obj
)
```

in T es el tipo de parámetro del método que este delegado encapsula.

obj es el parámetro del método que este delegado encapsula.



Delegado Func<T.Tresult>

- •Este delegado Encapsula un método que tiene un parámetro y devuelve un valor del tipo especificado por el parámetro *TResult*.
- •Este delegado puede usarse para representar un método que puede pasarse como parámetro sin declarar explícitamente ningún delegado personalizado. El método encapsulado debe corresponder a la firma del método definida por el delegado.



Sintaxis de Func<T.Tresult>

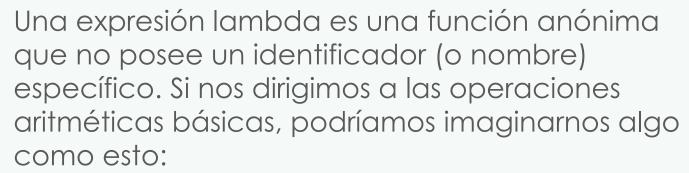
in T es el tipo de parámetro del método que este delegado encapsula.

Out TResult es el tipo del valor devuelto del método que este delegado encapsula.

T arg es el parámetro del método que este delegado encapsula.



Expresiones lambda



•Función convencional:

$$SumaCuadrados(x, y) = x \times x + y \times y$$

Función lambda:

$$(x, y) \mapsto x \times x + y \times y$$



Definición para C#

Una expresión lambda es un método anónimo (sin nombre) que se ha de declarar/definir (y asignar inmediatamente) sobre una **instancia de un delegado**.

Permiten una forma muy **compacta** de escribir funciones que puedan ser pasadas como argumentos para una evaluación posterior.



Definición para C#

Para crear una expresión lambda, hay que especificar parámetros de entrada (si existen) a la izquierda del operador => lambda, y colocar la expresión o bloque de instrucciones en el otro lado.

```
delegate int del(int i);
static void Main(string[] args)
{
   del myDelegate = x => x * x;
   int j = myDelegate(5); //j = 25
}
```



Definición para C#

```
class Test {
    delegate int DelegadoCuadrado(int numero);
    static int Cuadrado(int numero) {
        return numero * numero;
    static void Main() {
        //C# 1.0: Uso de delegados con un método estático
        DelegadoCuadrado dc1 = new DelegadoCuadrado(Cuadrado);
        Console.WriteLine("\t{0}", dc1(5));
        //C# 2.0: Delegado con método anónimo
        DelegadoCuadrado dc2 = delegate(int numero) {
            return numero * numero;
        };
        Console.WriteLine("\t{0}", dc2(7));
        // C# 3.0: Delegado con expresión lambda
        DelegadoCuadrado dc3 = x => x * x;
        Console.WriteLine("\t{0}", dc3(9));
        Console.ReadKey();
```

¿Qué es un evento?

- •Un evento es un mensaje que envía un objeto cuando ocurre una acción.
- •El objeto que **provoca** el evento se conoce como **remitente del evento**.
- •El objeto que **captura** el evento y responde a él se denomina **receptor del evento**.



Uso de eventos

- •Cuando ocurre **algo de interés**, los eventos habilitan una clase u objeto para **notificarlo** a otras clases u objetos.
- •Para consumir un evento en una aplicación, debe proporcionar un controlador de eventos (método de control de eventos) que ejecute la lógica del programa en respuesta al evento, y que registre el controlador de eventos en el origen del evento. Este proceso se denomina conexión de eventos.



Características de los eventos

- El remitente determina cuándo se produce un evento; los receptores determinan qué operación se realiza en respuesta al evento.
- Un evento puede tener varios receptores. Un receptor puede controlar varios eventos de varios remitentes.
- Los eventos se suelen usar para señalar acciones del usuario, como hacer clic en un botón o seleccionar un menú en interfaces gráficas de usuario.



Características de los eventos

- •Nunca se provocan eventos que no tienen receptores.
- •Si un evento tiene varios receptores, se invocan los controladores de eventos sincrónicamente cuando se produce el evento.
- •En la biblioteca de clases .NET Framework, los eventos se basan en el delegado EventHandler y en la clase base EventArgs.



Clase EventArgs

- •Representa la clase base para las clases que contienen datos de eventos y proporciona un valor para utilizar en los eventos que no incluyen datos de evento.
- •Para crear una clase de datos de eventos personalizada, se crea una clase que herede de la clase EventArgs y se proporcionan las propiedades para almacenar los datos necesarios.

