DELEGADOS EN C#

DELEGADOS

Definición

- Un delegado es un tipo especial de clase cuyos objetos pueden almacenar referencias a uno o más métodos con la misma signatura de tal manera que; a través del objeto, sea posible solicitar la ejecución en cadena de todos ellos.

Usos Principales

- Pasar una referencia a un método como parámetro.
- Ejecutar acciones asíncronas concurrentes tras un evento.
- Ejecutar el código de un método en otro hilo de forma paralela.

DELEGADOS

Sintaxis

Definición del tipo

```
<modificadores> delegate <tipoRetorno> <TipoDelegado>(<parám.formales>);
```

- Donde <TipoDelegado> será el nombre de la nueva que me servirá para definir objetos delegado. Mientras que <tipoRetorno> y <parámetros> se corresponderán, respectivamente, con el tipo del valor de retorno y la lista de parámetros de los métodos cuyos códigos puede almacenar en su interior los objetos de ese tipo delegado.

Instanciación y uso de objetos delegado

```
// Instanciación
TipoDelegado oDelegado = idMétodoConMismaSignatura;
// Equivale a ejecutar la antigua sintaxis.
// TipoDelegado oDelegado = new TipoDelegado(idMétodoConMismaSignatura);

// Uso
tipoRetorno resultado = objetoDelegado(<parám.reales>);
```

DELEGADOS

Ejemplo

```
using System;

class Principal
{
 // Definicón delegado
 public delegate double Operacion(double op1, double op2);

 // Definicón de métodos de clase con la misma signatura.
 static double Suma(double op1, double op2) { return op1 + op2; }

 static double Multiplica(double op1, double op2) { return op1 * op2; }

 // Recibe un objeto delegado del tipo Operación
 static double[] OperaArrays(
                     double[] ops1, double[] ops2,
                     Operacion operacion)
 {
    double[] resultados = new double[ops1.Length];
    for (int i = 0; i < resultados.Length; ++i)
        resultados[i] = operacion(ops1[i], ops2[i]);
    return resultados;
 }
```

DELEGADOS

Ejemplo (Continuación)

```
public static void Main()
{
    double[] ops1 = new double[] { 5, 4, 3, 2, 1 };
    double[] ops2 = new double[] { 1, 2, 3, 4, 5 };

    double[] sumas = OperaArrays(ops1, ops2, Suma);
    Console.Write("Sumas: ");
    foreach (double suma in sumas)
        Console.Write($"{suma} ");

    double[] multiplicaciones = OperaArrays(ops1, ops2, Multiplica);
    Console.Write("\nMultiplicaciones: ");
    foreach (double multiplicación in multiplicaciones)
        Console.Write($"{multiplicación} ");
}
}
```

DELEGADOS

Multidifusión De Delegados

- Se produce cuando un objeto delegado llama a más de un método cuando se invoca.
- Para encadenar un método / delegado en la multidifusión usaré el operador `+=`
- Para retirar un método / delegado de la multidifusión de llamadas usaré el operador `-=`

- Tiene sentido para métodos que no retornan nada (procedimientos), ya que si los delegados retornan algo como en el ejemplo anterior, se asignará el resultado de la última llamada.

DELEGADOS

Ejemplo Multidifusión De Delegados

```
class Principal
{
    public delegate void VerOperacion(int op1, int op2);

    public static void VerSuma(int op1, int op2) {
        Console.WriteLine($"{op1} + {op2} = {op1 + op2}");
    }

    public static void VerMultiplicacion(int op1, int op2) {
        Console.WriteLine($"{op1} * {op2} = {op1 * op2}");
    }

    public static void Main()
    {
        VerOperacion oDelegado = VerSuma;
        oDelegado += VerMultiplicacion;

        for (int i = 0; i < 10; ++i)
            oDelegado(i, i);
    }
}
```

DELEGADOS VS INTERFACES

Similitud

- En ambos casos llamamos a un método del que no sabemos su implementación.

Ejemplo

```
public delegate int Comparador<T>(T op1, T op2);

static void Odena<T>(List<T> l, Comparador<T> oDelegadoComparador) {
    …
    // En elgún momento llamaré al delegado.
    int comparacion = oDelegadoComparador(l[j], l[j+1]);
    …
}

static void Odena<T>(List<T> l, IComparer<T> oQueIMplementaComparer) {
    …
    // En elgún momento usaré la implementación del ineterfaz.
    int comparacion = oQueIMplementaComparer.Compare(l[j], l[j+1]);
    …
}
```

DELEGADOS VS INTERFACES

¿Cuándo Usar Uno U Otro?

Usar delegados cuando:

- Se utilice un modelo de diseño de eventos.
- Se prefiere a la hora de encapsular un método estático o de clase.
- El autor de las llamadas no tiene ninguna necesidad de obtener acceso a otras propiedades, métodos o interfaces en el objeto que implementa el método.
- Se desea conseguir una composición sencilla.
- Una clase puede necesitar más de una implementación del método.

Usar Interfaces cuando:

- Haya un grupo de métodos relacionados a los que se pueda llamar.
- Una clase sólo necesita una implementación del método.
- La clase que utiliza la interfaz deseará convertir esa interfaz en otra interfaz o tipos de clase.

DELEGADOS VS INTERFACES

Ejemplo

```
class Program {
    public static double MediaRaiz(double[] puntos) {
      double total = 0.0;
      for (int i = 0; i < puntos.Length; i++) {
          total += Math.Sqrt(puntos[i]);
      }
      return total / puntos.Length;
    }
    public static double MediaExponencial(double[] puntos)
    {
      double total = 0.0;
      for (int i = 0; i < puntos.Length; i++) {
          total += Math.Exp(puntos[i]);
      }
      return total / puntos.Length;
    }
    public static void Main() {
      double[] puntos = { 1, 2, 3, 4 };
      Console.WriteLine("Media raíces:" + MediaRaiz(puntos));
      Console.WriteLine("Media exponentes:" + MediaExponencial(puntos));
    }
}
```

DELEGADOS VS INTERFACES

- Viendo las dos funciones MediaRaiz y MediaExponencial hay cambios mínimos entre ambas.

- ¿No se podría programar una función "Media" genérica de forma que acepte la función real que le suministremos?

Solución Con Interfaces

```
public interface IFuncion {
    double Funcion(double valor);
}

public class MediaRaíz : IFuncion {
    public double Funcion(double valor) {
        return Math.Sqrt(valor);
    }
}

public class MediaExponente : IFuncion {
    public double Funcion(double valor){
        return Math.Exp(valor);
    }
}
```

DELEGADOS VS INTERFACES

Solución Con Interfaces (Continuación)

```
class Program {
    public static double Media(double[] puntos, IFuncion funcion) {
        double total = 0.0;
        for (int i = 0; i < puntos.Length; i++) {
            total += funcion.Funcion(puntos[i]);
        }
        return total / puntos.Length;
    }

    public static void Main() {
        double[] puntos = { 1, 2, 3, 4 };
        Console.WriteLine("Media raíces:" +
                          Media(puntos, new MediaRaíz()));
        Console.WriteLine("Media exponentes:" +
                          Media(puntos, new MediaExponente()));
    }
}
```

En este caso es preferible la solución con delegados poqeu vamos encapsular un método estático o de clase.

DELEGADOS VS INTERFACES

Solución Con Delegados

```
class Program {

    public delegate double Funcion(double valor);

    public static double Media(double[] puntos, Funcion funcion) {
        double total = 0.0;
        for (int i = 0; i < puntos.Length; i++) {
            total += funcion(puntos[i]);
        }
        return total / puntos.Length;
    }

    public static void Main() {

        double[] puntos = { 1, 2, 3, 4 };
        Console.WriteLine("Media raíces:" +
                          Media(puntos, Math.Sqrt));
        Console.WriteLine("Media exponentes:" +
                          Media(puntos, Math.Exp));
    }
}
```

EVENTOS EN C#

EVENTOS

Definición

- Un evento es un mensaje que envía un objeto cuando ocurre una acción.
- Es una forma de comunicación entre objetos.
- La acción podría ser debida a la interacción del usuario, como hacer clic en un botón, o podría proceder de cualquier otra lógica del programa, como el cambio del valor de una propiedad.
- Además, permiten a la clase informar de que es un tipo especial y así poder agruparlos y distinguirlos con el símbolo especial.

- Podremos diferenciar entre “2 tipos” de eventos dentro de C#:

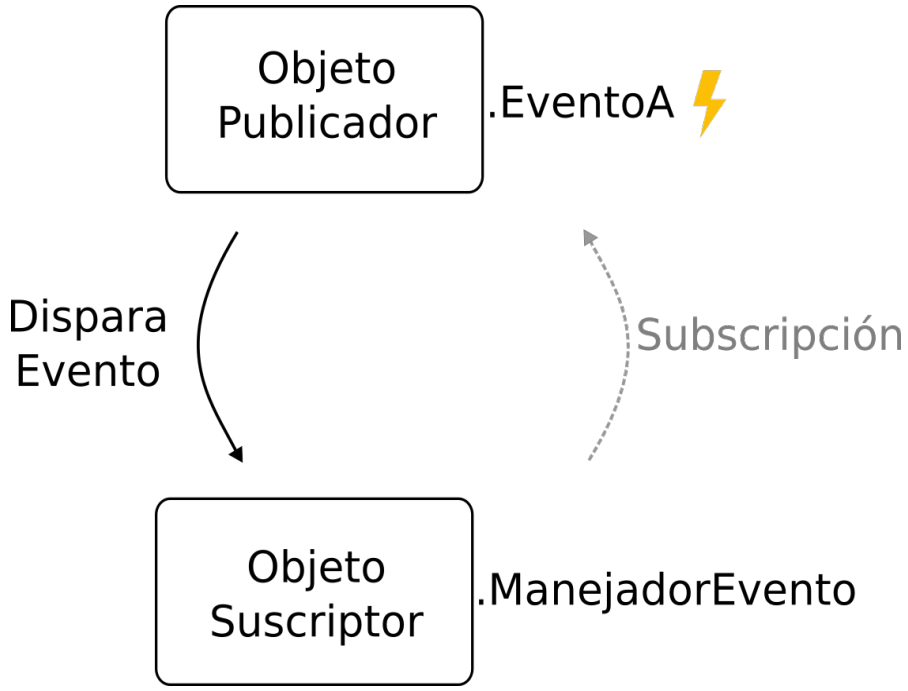
1. Los que me ayudarán a sincronizar varios hilos de ejecución.
2. Los que me indicarán una determinada ocurrencia definida por el usuario, desencadenando una operación asíncrona.

Nosotros en esta sección vamos a tratar los segundos, asociados a un delegado encargado de ejecutar esa operación asíncrona.

EVENTOS

Nomenclatura Publicador - Suscriptor

- Los métodos asociados al delegado que se ejecutan al producirse un evento se denominan manejadores o controladores.
- El objeto que provoca el evento se conoce como emisor o piublicador del evento y es el que contiene el tipo delegado.
- El objeto que maneja el evento se conoce como receptor o suscriptor del evento. Debe ser un objeto diferente al objeto emisor y es el que contiene los controladores.



Sintaxis

```
<modificadores> event <TipoDelegado> idEvento;
```

EVENTOS

Suscripción Y Desencadenamiento De Eventos

- Supngamos la siguiente definición…

```
class Publicador {
 public delegate void ManejadorDeEvento();
 public event ManejadorDeEvento IdEvento;
 …
}
```

- Para asociarlo a un manejador con la signatura, usaremos el operador +=

```
class Suscritor {
 public void  ManejadorDeEventoDelSuscriptor() { … }
 …
}
// Suscripción del oSuscriptor al evento IdEvento
oPublicador.IdEvento += oSuscriptor.ManejadorDeEventoDelSuscriptor();
```

- Solo se desencadenará si tenemos algún manejador o controlador asociado.

```
class Publicador {
 // Desencadenar el evento.
 if (IdEvento) IdEvento.Invoke();
 // O También... if (IdEvento) IdEvento();
}
```

EVENTOS

Veamos Un Ejemplo Un Poco Más Elaborado

```
class Publicador {

    const int LIMITE = 9;
    private int contador;
    public delegate void ManejadorDeEvento<T>(T publicador);
    public event ManejadorDeEvento<Publicador> LímiteSuperado;

    public Publicador() { contador = 0; }

    public virtual void Incrementa() {
        contador++;
        if (contador > LIMITE && LímiteSuperado != null)
            LímiteSuperado.Invoke(this);
    }

    public override string ToString() {
        return $"Contador = {contador}";
    }
}
```

EVENTOS

```
class Suscriptor {
    // Manejador que gestiona el evento/mensaje del publicador.
    public void ManejadorLímiteSuperadoPorPublicador(
                                                Publicador publicador) {
        Console.WriteLine("P ha superado el límite.");
    }
}

class Principal {
    public static void Main() {
        Publicador p = new Publicador();
        Suscriptor s = new Suscriptor();

        p.LímiteSuperado += s.ManejadorLímiteSuperadoPorPublicador;

        for (int i = 1; i < 12; i++) {
            p.Incrementa();
            Console.WriteLine(p);
        }
    }
}
```

- Cada vez que se llame a incrementa y el límite esté superado. EL suscriptor será informado a través de su manejador y de forma asíncrona, de que esto ha sucedido.

DELEGADOS PARAMETRIZADOS EN C#

DELEGADOS GENÉRICOS

Un delegado puede estar parametrizado:

- Esto me permitirá usar un tipo delegado para las signaturas de métodos más comunes que se me pueden dar.

```
public delegate bool Predicado<T>(T p);

public static bool EsPar(int valor) {

     return (valor % 2 == 0);
}

static void Main() {

    Predicado<int> predicado = EsPar;
    Console.WriteLine(predicado(4));
}
```

DELEGADOS GENÉRICOS PREDEFINIDOS EN LA BCL - I

En .NET ya han tenido en cuenta esto y ya vienen predefinidos un gran número de tipos delegados genéricos que podré usar.

Delegate Bool Predicate<in T>(T Obj)

- Me sirve para definir predicados o métodos que reciben un objeto y me indican si cumple una determinada condición.

```
public static bool EsImpar(int valor) {

     return (valor % 2 != 0);
}

static void Main() {

    List<int> valores = new List<int> { 2, 6, 3, 8, 2 };

    Predicate<int> predicado = EsImpar;
    Console.WriteLine(valores.Find(predicado));
}
```

DELEGADOS GENÉRICOS PREDEFINIDOS EN LA BCL - II

Delegate Void Action()

Delegate Void Action<in T1, In T2, …, In T16>(T Obj1, T2 Obj2, …, T16 Obj16)

- Me permite definir delegados que funcionan como procedimientos, esto es, pueden recibir hasta de 0 a 16 parámetros genéricos y no retornan nada.

```
public static void Muestra(int valor) {

    Console.WriteLine($"{valor:D2}");
}

static void Main() {

    List<int> valores = new List<int> { 2, 6, 3, 8, 2 };

    Action<int> muestra = Muestra;
    valores.ForEach(muestra);
}
```

DELEGADOS GENÉRICOS PREDEFINIDOS EN LA BCL - II

Delegate R Func<out R>()

Delegate R Func<in T1, …, In T16, Out R>(T Obj1, …, T16 Obj16)

- Me permite definir delegados que funcionan como funciones, esto es, pueden recibir hasta de 0 a 16 parámetros genéricos y el último por la derecha será el tipo de retorno de la función.

```
public static string ACadena(int valor) {

     return $"<{valor:D2}>";
}

static void Main(){

    List<int> valores = new List<int> { 9, 6, 3, 8};

    Func<int, string> minimoACadena = ACadena;
    Console.WriteLine(valores.Min<int, string>(minimoACadena));
}
```