Delegados

Delegados

- Un Delegado es un tipo de dato
- Encapsula métodos y guarda la instancia de a quién pertenece ese método
- El nombre del delegado define su tipo
- Similar a los punteros a funciones en C

```
public delegate tiporetorno NombreDelegado(paramquerecibe);
```

```
using System;

namespace Delegados
{

    public delegate void MensajeDelegate();
```

- Al instanciarlos podemos asociar su instancia con cualquier método que tenga una firma compatible

- Los delegados permiten pasar los métodos como parámetros

Es posible usar el operador += para agregar más métodos a la lista y el operador -= para remover métodos de la lista

```
namespace Delegados
{
    public delegate void MensajeDelegate();

    0 referencias
    class Program
    {
        1 referencia
        static void MensajeUno() { Console.WriteLine("Mensaje 1"); }
        1 referencia
        static void MensajeDos() { Console.WriteLine("Mensaje 2"); }

        0 referencias
        static void Main(string[] args)
        {
            MensajeDelegate delMensaje;
            delMensaje = MensajeUno;
            delMensaje += MensajeDos;
        }
    }
}
```

Para vaciarlo podemos asignarlo a null y para saber si tiene algo cargado preguntamos si != de null

```
static void Main(string[] args)
{
    MensajeDelegate delMensaje;
    delMensaje = MensajeUno;
    delMensaje += MensajeDos;

    if(delMensaje != null)
        Console.WriteLine("Tiene metodos");
    else
        Console.WriteLine("Esta vacio");

    //remover un metodo
    delMensaje -= MensajeDos;

    //vaciarlo completamente
    delMensaje = null;
}
```

Los métodos de la lista serán invocados uno por uno secuencialmente, aunque no siempre en el orden en que fueron agregados a la lista

```
class Program
{
    1 referencia
    static void MensajeUno()...
    1 referencia
    static void MensajeDos()...

    0 referencias
    static void Main(string[] args)
    {
        MensajeDelegate delMensaje;
        delMensaje = MensajeUno;
        delMensaje += MensajeDos;

        //forma 1
        delMensaje.Invoke();

        //forma 2
        delMensaje();

    }
}
```

Action<T>

Action<T>, es un delegado genérico que retorna void. Es decir, un Action<T> nos permite crear un delegado sin tener que declarar el tipo

```
using System;

namespace Delegados
{
    0 referencias
    class Program
    {
        1 referencia
        static void MensajeUno(string msj)
        { Console.WriteLine("mensaje 1: " + msj); }
        1 referencia
        static void MensajeDos(string msj)
        { Console.WriteLine("mensaje 2: " + msj); }

        0 referencias
        static void Main(string[] args)
        {
            Action<string> delegado1;
            delegado1 = MensajeUno;
            delegado1 += MensajeDos;

            delegado1.Invoke("Hola mundo");
        }
    }
}
```

Existe, también, un Action no-genérico que no recibe parámetros

```
using System;

namespace Delegados
{
    0 referencias
    class Program
    {
        1 referencia
        static void MensajeUno() { Console.WriteLine("Mensaje 1"); }
        1 referencia
        static void MensajeDos() { Console.WriteLine("Mensaje 2"); }

        0 referencias
        static void Main(string[] args)
        {

            Action delegado1;
            delegado1 = MensajeUno;
            delegado1 += MensajeDos;

            delegado1.Invoke();

        }
    }
}
```

Func<T>

Delegado genérico, donde la variable genérica es el tipo de valor de retorno

```
0 referencias
class Program
{
    1 referencia
    static string MensajeUno()
    { return "mensaje : 1"; }
    1 referencia
    static string MensajeDos()
    { return "mensaje : 2"; }

    0 referencias
    static void Main(string[] args)
    {
        Func<string> delMensajes;
        delMensajes = MensajeUno;
        delMensajes += MensajeDos;

        string msj = delMensajes.Invoke();
        Console.WriteLine(msj);
    }
}
```

Variante del Delegado Func

Func<in T, out TResult>

Esta variante es un delegado genérico que permite especificar cuál va a ser su retorno y cuál el parámetro de ingreso.

T es el tipo de parámetro que ingresa, y TResult es el tipo de parámetro que devuelve

```
0 referencias
class Program
{
    1 referencia
    static string Mensaje(int nroMensaje)
    { return "mensaje : " + nroMensaje; }

    0 referencias
    static void Main(string[] args)
    {
        Func<int,string> delMensajes;
        delMensajes = Mensaje;

        string msj = delMensajes.Invoke(1);
        Console.WriteLine(msj);
    }
}
```

Tanto el Action como el Func, admiten cualquier tipo y cantidad de parámetros

El delegado se adapta a lo que le pasemos por parámetro

En los Func el último parámetro es el tipo que devuelve

DELEGADO	EQUIVALENTE	DELEGADO	EQUIVALENTE
Action<T>	void Method(T param1)	Func<T>	T Method()
Action<T, A>	void Method(T param1, A param2)	Func<A, T>	T Method(A param1)
Action<T, A, B>	void Method(T param1, A param2, B param3)	Func<B, A, T>	T Method(A param1, B param2)