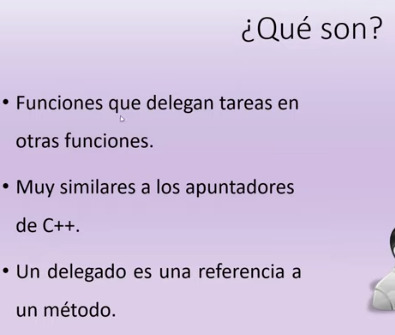
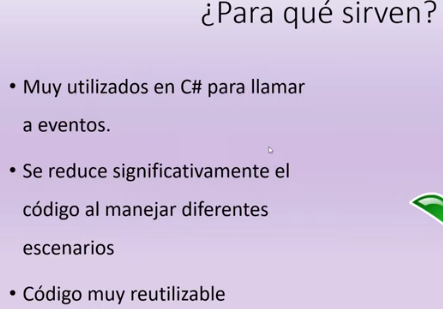
Delegados, predicados y lambdas

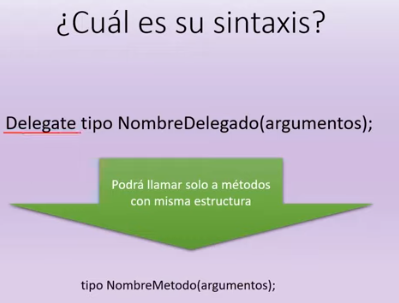
Delegados:



Son funciones que delegan tareas en otras funciones.

Muy utilizados para llamar a eventos.





Si llama a un método void, el delegado debe ser void. Los argumentos también deben ser los mismos, en cuanto a estructura.

Es capaz de llamar a métodos que se encuentran en otros ficheros.

class Program

{

static void Main(string[] args)

{

//creación del objeto delegado apuntando a mensajebienvenida (nombre de clase) + SaludoBienvenida (nombre del método)

ObjetoDelegado ElDelegado = new ObjetoDelegado(MensajeBienvenida.SaludoBienvenida);

//Utilización del delegado para llamar al método saludo bienvenida

ElDelegado();

ElDelegado = new ObjetoDelegado(MensajeDespedida.SaludoDespedida);

ElDelegado();

}

//definición del objeto delegado

delegate void ObjetoDelegado();

}

class MensajeBienvenida

{

public static void SaludoBienvenida()

{

Console.WriteLine("Hola, acabo de llegar ¿Qué tal?");

}

}

class MensajeDespedida

{

public static void SaludoDespedida()

{

Console.WriteLine("Adios, me retiro");

}

}

}

Esto funciona y devuelve los 2 mensajes

Ahora, agrega a los métodos un string como parámetro.

class MensajeBienvenida

{

public static void SaludoBienvenida(string msj)

{

Console.WriteLine("Mensaje de bienvenida {0}", msj);

}

Al hacer eso, va a saltar error cuando se construya el objeto, porque ya no son iguales. Así que corrige eso:

class Program

{

static void Main(string[] args)

{

//creación del objeto delegado apuntando a mensajebienvenida

ObjetoDelegado ElDelegado = new ObjetoDelegado(MensajeBienvenida.SaludoBienvenida);

//Utilización del delegado para llamar al método saludo bienvenida

ElDelegado("Soy lo mas, adoreme señora");

ElDelegado = new ObjetoDelegado(MensajeDespedida.SaludoDespedida);

ElDelegado("Nos vemos en Disniwor");

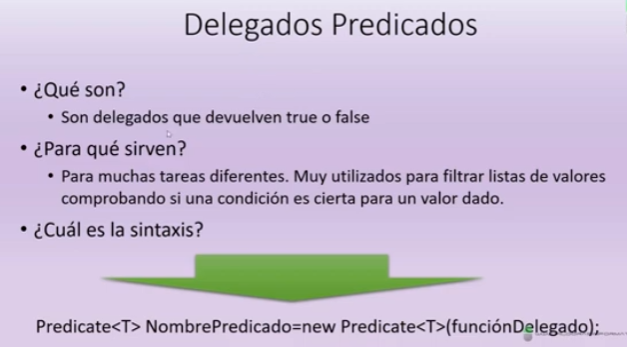
}

//definición del objeto delegado

delegate void ObjetoDelegado(string mensaje);

}

Predicados



class Program

{

static void Main(string[] args)

{

List <int> ListaNumeros = new List<int>();

ListaNumeros.AddRange(new int[] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 });

//ahí mismo crea un array con esos valores. Con Addrange se los pasa.

//Declaración de delegado predicado

Predicate<int> ElDelegadoPred = new Predicate<int>(DamePares);

//Va a devolver una lista, por lo que hay que crearla para poder almacenar el return.

List<int> NumPares = ListaNumeros.FindAll(ElDelegadoPred);

foreach (int num in NumPares)

{

Console.WriteLine(num);

}

}

static bool DamePares (int numero)

{

if (numero % 2 == 0) return true;

else return false;

}

Ahí va otro:

namespace Delegados\_Predicados\_Lambdas

{

class Program

{

static void Main(string[] args)

{

List<Personas> gente = new List<Personas>();

Personas P1 = new Personas();

P1.Nombre = "Juan"; P1.Edad = 18;

Personas P2 = new Personas();

P2.Nombre = "Roberto"; P2.Edad = 38;

Personas P3 = new Personas();

P3.Nombre = "Ricardo"; P3.Edad = 41;

gente.AddRange(new Personas[] { P1, P2, P3 });

Predicate<Personas> PredicadoPersonas = new Predicate<Personas>(ExisteJuan);

bool existe = gente.Exists(PredicadoPersonas);

if (existe==true) Console.WriteLine("Hay al menos una persona llamada Juan");

else Console.WriteLine("No hay nadie llamado Juan");

}

static bool ExisteJuan(Personas pesrsona)

//ahí está recibiendo un objeto de tipo Personas, llamado persona

{

if (pesrsona.Nombre == "Juan") return true;

else return false;

}

}

class Personas

{

private string nombre;

private int edad;

public string Nombre { get => nombre; set => nombre = value; }

public int Edad { get => edad; set => edad = value; }

}

}

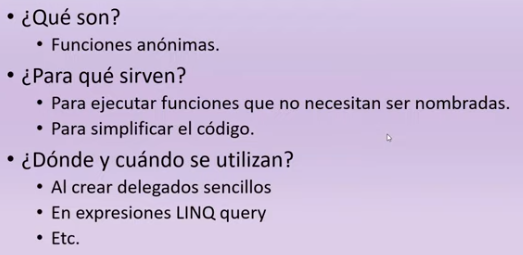
Lambda

Al principio suele ser difícil de leer.

Son funciones anónimas. No tienen nombre.

Simplifican el código, permitiendo ejecutar funciones sencillas.

Se las utiliza al crear delegados sencillos y al utilizar expresiones LINQ query, mas que nada.





Se reduce una función a algo mas simple.

static void Main(string[] args)

{

/\* Personas P1 = new Personas();

P1.Nombre = "Juan"; P1.Edad = 18;

Personas P2 = new Personas();

P2.Nombre = "Roberto"; P2.Edad = 38;\*/

//uso del delegado

OperacionesMatematicas operacion = new OperacionesMatematicas(Cuadrado);

Console.WriteLine(operacion(4));

}

public delegate int OperacionesMatematicas(int numero);

/\*

Prescindiendo del lambda, se podría poner la siguiente función y funciona joya

\* public static int Cuadrado (int num)

{

return num \* num;

}\*/

public static int Cuadrado(int num)

{

return num \* num;

}

}

Del delegado se pasa el int a la función. No hace falta que el nombre de la variable sea el mismo.

Luego, usa el lambda:

{

OperacionesMatematicas operacion = new OperacionesMatematicas(num=> num\*num);

Console.WriteLine(operacion(4));

}

public delegate int OperacionesMatematicas(int numero);

Función lambda con 2 parámetros.

{

OperacionesMatematicas operacion = new OperacionesMatematicas(Suma);

Console.WriteLine(operacion(4,7));

}

public delegate int OperacionesMatematicas(int numero1, int numero2);

public static int Suma (int Num1, int Num2)

{

return Num1 + Num2;

}

Esta es la versión sin labda, ahora va con lambda:

OperacionesMatematicas operacion = new OperacionesMatematicas((num1,num2)=> num1+num2);

Console.WriteLine(operacion(4,7));

}

public delegate int OperacionesMatematicas(int numero1, int numero2);

Ahí se ve que los parámetros, ya que son 2, van entre paréntesis.

static void Main(string[] args)

{

Personas P1 = new Personas();

P1.Nombre = "Juan"; P1.Edad = 18;

Personas P2 = new Personas();

P2.Nombre = "Roberto"; P2.Edad = 38;

List<int> numeros = new List<int> { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

List<int> NumerosPares = numeros.FindAll(i => i % 2 == 0); //Lambda. A cada valor se le realiza la operación.

foreach (int numero in NumerosPares) Console.WriteLine(numero);

//también podría hacerse así, con el lambda

NumerosPares.ForEach(numero => Console.WriteLine(numero));

}

Si a la derecha del operador lambda hay mas de 2 operaciones se abren llaves, y separando las operaciones, va ; También va ; tras la segunda operación, y obvio, ; al final de todo.

NumerosPares.ForEach(numero => { Console.WriteLine("El número par es: "); Console.WriteLine(numero); });

Acá otro, utilizando objetos:

class Program

{

static void Main(string[] args)

{

Personas P1 = new Personas();

P1.Nombre = "Juan"; P1.Edad = 18;

Personas P2 = new Personas();

P2.Nombre = "Roberto"; P2.Edad = 38;

ComparaPersonas comparaEdad = (per1, per2) => per1 == per2;

Console.WriteLine(comparaEdad(P1.Edad, P2.Edad));

}

public delegate bool ComparaPersonas(int num1, int num2);

}

class Personas

{

private string nombre;

private int edad;

public string Nombre { get => nombre; set => nombre = value; }

public int Edad { get => edad; set => edad = value; }

}