Parámetros por valor y referencia

Índice

1 - Parámetros por valor

Cuando tenemos un método le pasamos un parámetro, en verdad no estamos mandando ese parámetro, estamos mandado una copia de este, lo que significa, que fuera del método, el valor de la variable seguirá siendo el mismo. Incluso si modificamos su valor internamente.

```
static void Main(string[] args)
{
    int valorActual = 10;

    Actualizar(valorActual);
    Console.WriteLine($"el valor es: {valorActual}");//imprime 10

    Console.ReadKey();
}

//Comportamiento normal.
public static void Actualizar(int valor)
{
    valor += 5;
    Console.WriteLine($"el valor es: {valor}"); //imprime 15
}
```

`valorActual` sigue valiendo 10 después de finalizar el método `Actualizar()` ya que el valor únicamente se modifica para ese método.

2 - Parámetros por referencia

Puede llegar el momento de que necesitemos actualizar el valor de los datos. Para ello pasaremos los parámetros por referencia, y lo haremos utilizando las palabras

reservadas `ref` y `out`. Cuando utilizamos `ref` y `out` no almacenamos en la variable el valor de esta, sino que almacenamos la posición de memoria a la que apunta por lo que, si modificamos el valor, también cambiamos el valor de la original.

2.1 - Palabra reservada ref

Utilizar `ref` significa que vas a pasar el valor por referencia a un método, lo que implica que el valor va a cambiar dentro del método. Además, si deseamos utilizar `ref` la variable tiene que estar inicializada con anterioridad.

Finalmente debemos indicar tanto en el método como en la llamada al método que vamos a pasar un valor por referencia.

```
static void Main(string[] args)
{
    int valorActual = 10;

    ActualizarRef(ref valorActual); //pasar por referencia
    Console.WriteLine($"el valor es: {valorActual}"); // imprime 12


    Console.ReadKey();
}

//Actualizar por referencia
public static void ActualizarRef(ref int valor)
{
    valor += 2;
}
```

2.2 - Palabra reservada out

Utilizar `out` significa que la asignación del valor de esa variable está dentro del método al que se llama. No es necesario inicializar el valor de la variable, aunque si debemos instanciarla.

```
static void Main(string[] args)
{

    int valorActual;
    ActualizarOut(out valorActual);
    Console.WriteLine($"el valor es: {valorActual}"); // imprime 13

    Console.ReadKey();
}


//Crear utilizando out
```

```
public static void ActualizarOut(out int valor)
{
    valor = 13;
}
```

2.3

La or

lo c

```
public static void ActualizarIn(in int valor)
{
    valor += 5; //Da Error
    Console.WriteLine($"el valor es: {valor}");
}
```

3 - El caso especial de objetos

Cuando pasamos un Objeto/Tipo creado por nosotros mismos, este siempre se pasa por referencia. Con lo que, si lo actualizamos dentro de la función, se actualizara fuera de ella.

```
class Program
{
    static void Main(string[] args)
    {

        ObjEjemplo ejemploValor = new ObjEjemplo(10);
        ActualizarObj(ejemploValor);
        Console.WriteLine($"el valor es: {ejemploValor.Entero}"); // imprime 25

        Console.ReadKey();
    }

    public static void ActualizarObj(ObjEjemplo obj)
    {
        obj.Entero = 25;
    }

}

public class ObjEjemplo
{
    public int Entero { get; set; }

    public ObjEjemplo(int entero)
```

```
    {
        Entero = entero;
    }
}
```

4 - I

- De
  va
- No

5 - (

Util

mé

Des

de

6 - (

El c