Métodos y parámetros

Notas generales

- Uso de métodos
- Uso de parámetros
- Uso de métodos sobrecargados
- Práctica – Creación y uso de métodos

Uso de métodos

- Definición de métodos
- Llamadas a métodos
- Uso de la instrucción return
- Uso de variables locales
- Devolución de valores

Definición de métodos

- **Main es un método**
  - Para definir métodos propios se usa la misma sintaxis

```
using System;

class ExampleClass
{
    static void ExampleMethod( )
    {
        Console.WriteLine("ExampleMethod");
    }
    static void Main( )
    {
        // ...
    }
}
```

Llamadas a métodos

- **Una vez definido un método, se puede:**
  - Llamar a un método desde dentro de la misma clase

    Se usa el nombre del método seguido de una lista de parámetros entre paréntesis
  - Llamar a un método que está en una clase diferente

    Hay que indicar al compilador cuál es la clase que contiene el método que se desea llamar

    El método llamado se debe declarar con la palabra clave **public**
  - Usar llamadas anidadas

    Unos métodos pueden hacer llamadas a otros, que a su vez pueden llamar a otros métodos, y así sucesivamente

Uso de la instrucción return

- Return inmediato
- Return con una instrucción condicional

```
static void ExampleMethod( )
{
    int numBeans;
    //...

    Console.WriteLine("Hello");
    if (numBeans < 10)
        return;
    Console.WriteLine("World");
}
```

Uso de variables locales

- **Variables locales**
  - Se crean cuando comienza el método
  - Son privadas para el método
  - Se destruyen a la salida
- **Variables compartidas**
  - Para compartir se utilizan variables de clase
- **Conflictos de ámbito**
  - El compilador no avisa si hay conflictos entre nombres locales y de clase

Devolución de valores

- **El método se debe declarar con un tipo que no sea void**
- **Se añade una instrucción return con una expresión**
  - Fija el valor de retorno
  - Se devuelve al llamador
- **Los métodos que no son void deben devolver un valor**

```
static int DosMasDos( ) {
    int a,b;
    a = 2;
    b = 2;
    return a + b;
}
```

```
int x;
x = DosMasDos( );
Console.WriteLine(x);
```

Uso de parámetros

- Declaración y llamadas a parámetros
- Mecanismos de paso de parámetros
- Paso por valor
- Paso por referencia
- Parámetros de salida
- Uso de listas de parámetros de longitud variable
- Normas para el paso de parámetros
- Uso de métodos recursivos

Declaración y llamadas a parámetros

- **Declaración de parámetros**
  - Se ponen entre paréntesis después del nombre del método
  - Se definen el tipo y el nombre de cada parámetro
- **Llamadas a métodos con parámetros**
  - Un valor para cada parámetro

```
static void  MethodWithParameters(int n, string y)
{ ... }

MethodWithParameters(2, "Hola, mundo");
```

Mecanismos de paso de parámetros

- **Tres maneras de pasar parámetros**

entrada	Paso por valor
entrada salida	Paso por referencia
salida	Parámetros de salida

Paso por valor

- **Mecanismo predeterminado para el paso de parámetros:**
  - Se copia el valor del parámetro
  - Se puede cambiar la variable dentro del método
  - No afecta al valor fuera del método
  - El parámetro debe ser de un tipo igual o compatible

```
static void SumaUno(int x)
{
      x++; // Incrementar x
}
static void Main( )
{
      int k = 6;
      SumaUno(k);
      Console.WriteLine(k); // Muestra el valor 6, no 7
}
```

Paso por referencia

- **¿Qué son los parámetros referencia?**
  - Una referencia a una posición de memoria
- **Uso de parámetros referencia**
  - Se usa la palabra clave **ref** en la declaración y las llamadas al método
  - Los tipos y valores de variables deben coincidir
  - Los cambios hechos en el método afectan al llamador
  - Hay que asignar un valor al parámetro antes de la llamada al método

Parámetros de salida

- **¿Qué son los parámetros de salida?**
  - Pasan valores hacia fuera, pero no hacia dentro
- **Uso de parámetros de salida**
  - Como **ref**, pero no se pasan valores al método
  - Se usa la palabra clave **out** en la declaración y las llamadas al método

```
static void OutDemo(out int p)
{
      // ...
}
int n;
OutDemo(out n);
```

Uso de listas de parámetros de longitud variable

- Se usa la palabra clave params
- Se declara como tabla al final de la lista de parámetros
- Siempre paso por valor

```
static long AddList(params long[ ] v)
{
    long total, i;
    for (i = 0, total = 0; i < v.Length; i++)
        total += v[i];
    return total;
}
static void Main( )
{
    long x = AddList(63,21,84);
}
```

Normas para el paso de parámetros

- **Mecanismos**
  - El paso por valor es el más habitual
  - El valor de retorno del método es útil para un solo valor
  - **ref** y/o **out** son útiles para más de un valor de retorno
  - **ref** sólo se usa si los datos se pasan en ambos sentidos
- **Eficiencia**
  - El paso por valor suele ser el más eficaz

Uso de métodos sobrecargados

- Declaración de métodos sobrecargados
- Signaturas de métodos
- Uso de métodos sobrecargados

Declaración de métodos sobrecargados

- **Métodos que comparten un nombre en una clase**
  - Se distinguen examinando la lista de parámetros

```
class OverloadingExample
{
    static int Suma(int a, int b)
    {
        return a + b;
    }
    static int Suma(int a, int b, int c)
    {
        return a + b + c;
    }
    static void Main( )
    {
        Console.WriteLine(Suma(1,2) + Suma(1,2,3));
    }
}
```

Signaturas de métodos

- Las signaturas de métodos deben ser únicas dentro de una clase
- Definición de signatura

Forman la definición de la signatura

- Nombre del método
- Tipo de parámetro
- Modificador

No afectan a la signatura

- Nombre de parámetro
- Tipo de retorno de método

Uso de métodos sobrecargados

- **Conviene usar métodos sobrecargados si:**
  - Hay métodos similares que requieren parámetros diferentes
  - Se quiere añadir funcionalidad al código existente
- **No hay que abusar, ya que:**
  - Son difíciles de depurar
  - Son difíciles de mantener

Práctica - Creación y uso de métodos