UNIVERSIDAD AUTÓNOMA DE SINALOA

Facultad de Informática Culiacán

Programación Modular en C#

Instructor:

MC. Gerardo Gálvez Gámez

gerardo.galvez@uas.edu.mx

Septiembre de 2017

Métodos y Parámetros

C#

Notas Generales

Es recomendable descomponer la lógica del programa en unidades funcionales, ya que facilita la reutilización del código.

1. Uso de métodos.
2. Uso de parámetros.
3. Uso de métodos sobrecargados.

Módulos en C#• FIUAS

Uso de Métodos

Uno de los principios básicos del diseño de aplicaciones es que deben estar divididas en unidades funcionales, ya que las secciones pequeñas de código son más fáciles de entender, diseñar, desarrollar y depurar.

La división de una aplicación en unidades funcionales permite también la reutilización de componentes funcionales en toda la aplicación.

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

Uso de Métodos

- Una aplicación C# está estructurada en clases que contienen bloques de código con nombre llamados métodos.
- Un *método* es un miembro de una clase que lleva a cabo una acción o calcula un valor.
- Se utilizan para estructurar el código de un programa.

Módulos en C#• FIUAS

Definición de Métodos

Un método es una serie de instrucciones C# que han sido agrupadas bajo un nombre determinado.

Ejemplo:

- Main es un método

Para definir métodos propios se usa la siguiente sintaxis:

```
static void MethodName(Lista de parámetros )
{
        cuerpo del método
}
```

Creación de Métodos

Cuando se crea un método hay que especificar lo siguiente:

- **Nombre**
  - Un método no puede tener el mismo nombre que una variable, una constante o cualquier otro elemento que no sea un código y haya sido declarado en la clase. El nombre del método puede ser cualquier identificador permitido de C# y distingue entre mayúsculas y minúsculas.
- **Lista de parámetros**
  - A continuación del nombre del método viene una lista de parámetros para el método. Esta lista aparece entre paréntesis que deben estar presentes aunque no hay ningún parámetro, como se ve en los ejemplos de la transparencia.
- **Cuerpo del método**
  - Después de los paréntesis viene el cuerpo del método, que debe estar entre llaves ({ y }) aunque no contenga más que una instrucción.

Módulos en C# • FIUAS

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 declararse 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

La instrucción **return** se puede emplear para hacer que un método se devuelva inmediatamente al llamador. Sin una instrucción **return**, lo normal es que la ejecución se devuelva al llamador cuando se alcance la última instrucción del método.

- Return inmediato
- Return con una instrucción condicional

```
static void ExampleMethod( )
{
   int Numero;
   Console.WriteLine("Hello");
   if (Numero < 10)
   {
      return;
   }
   Console.WriteLine("World");
}
```


Return inmediato

Por defecto, un método es devuelto a su llamador cuando se llega al final de la última instrucción del bloque de código.

La instrucción **return** se utiliza cuando se quiere que un método sea devuelto inmediatamente al llamador.

Ejemplo:

```
static void ExampleMethod( )
 {
    Console.WriteLine("Hola");
    return;
    Console.WriteLine("mundo");//código inalcanzable
 }
```

Return con una instrucción condicional

Es mucho más habitual, y mucho más útil, utilizar la instrucción **return** como parte de una instrucción condicional como **if** o **switch**.

Esto permite que un método sea devuelto al llamador si se cumple cierta condición.

Ejemplo:

```
static void ExampleMethod( )
{
   int Numero;
   Console.WriteLine("Hello");
   if (Numero < 10)
   {
      return;
   }
   Console.WriteLine("World");
}
```

Módulos en C#• FIUAS

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.

Variables Locales

```
static void MethodWithLocals( )
 {
    int x = 1; // Variable con valor inicial
   ulong y;
   string z;
    ...
}
```

Módulos en C#• FIUAS

Variables Compartidas

```
class CallCounter_Good
 {
   static int nCount;

   static void Init( )
   {
      nCount = 0;
    }
    static void CountCalls( )
    {
      ++ nCount;
      Console.Write("Método llamado " + nCuenta + " veces.");
    }
   static void Main( )
    {
      Init( );
       CountCalls( );
       CountCalls( );
    }
}
```

Devolución de valores

- El método se debe declarar con un tipo que no sea void.
- Se añade una instrucción return con una expresión donde:
  - Se 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;
}
```

```
static void Main( )
{
    int x;
    x = DosMasDos( );
    Console.WriteLine(x);
}
```


Los métodos que no son void

Deben devolver valores:

1. **Si se declara un método con un tipo distinto de void, es obligatorio añadir al menos una instrucción return.**

   El compilador intenta comprobar que cada uno de estos métodos devuelve siempre un valor al método de llamada.

2. **Si detecta que un método que no es void no incluye ninguna instrucción return, el compilador mostrará el siguiente mensaje de error: "No todas las rutas de código devuelven un valor."**

   Este mensaje también aparecerá si el compilador detecta que es posible ejecutar un método que no es void sin devolver un valor.

Codificación de Algoritmos al Lenguaje.

- Elabore un algoritmo que solicite al usuario dos números reales, calcule la suma e imprima el resultado en pantalla, utilice variables compartidas y métodos para leer, calcular e imprimir.

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");
```

Módulos en C#• FIUAS

Llamadas a métodos con parámetros

El código de llamada debe indicar los valores de los parámetros en la llamada al método.

Ejemplos:

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

int p = 7;
string s = "Mensaje de prueba";
MethodWithParameters(p, s);
```

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.
  - A diferencia de un parámetro valor, un parámetro referencia no crea una nueva ubicación de almacenamiento. Por el contrario, un parámetro referencia representa la misma posición de memoria que la variable indicada en la llamada al método.

Módulos en C#• FIUAS

Paso por referencia

- 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.

Ejemplo

```
static void ShowReference(ref int nId, ref long nCount)
{
  // ...
}
```

La palabra clave **ref** afecta únicamente al parámetro al que antecede, no a toda la lista de parámetros. En el siguiente método, *nId* se pasa por referencia pero *longVar* se pasa por valor:

```
static void OneRefOneVal(ref int nId, long longVar)
 {
   //
    ...
 }
```

Módulos en C#• FIUAS

En la llamada a un método

Los parámetros referencia se indican utilizando la palabra clave **ref** seguida de una variable.

El valor indicado en la llamada al método debe ser exactamente igual al tipo en la definición del método, y además debe ser una variable, no una constante ni una expresión calculada.

Ejemplo:

```
int x;
long q;
ShowReference(ref x, ref q);
```

Codificación de Algoritmos al Lenguaje.

- Elabore un algoritmo que solicite al usuario dos números reales, calcule la suma e imprima el resultado en pantalla, utilice métodos que reciban como parámetros por referencia los datos a leer y por valor los datos a calcular e imprimir.

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);
```

Parámetros de salida

Transfieren datos fuera del método en lugar de al método.

Al igual que u parámetro referencia, un parámetro de salida es una referencia a una ubicación de almacenamiento indicada por el llamador.

Sin embargo, no es necesario asignar un valor a la variable indicada para el parámetro **out** antes de hacer la llamada, y el método asume que el parámetro no ha sido inicializado.

Los parámetros de salida son útiles cuando se quiere devolver valores de un método por medio de un parámetro sin tener que asignar a éste un valor inicial.

Codificación de Algoritmos al Lenguaje.

- Elabore un algoritmo que solicite al usuario dos números reales, calcule la suma e imprima el resultado en pantalla, utilice métodos que reciban como parámetros de salida los datos a leer y por valor los datos a calcular e imprimir.

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


Paso de valores

Hay dos maneras de pasar valores al parámetro **params** cuando se hace una llamada a un método con un parámetro de longitud variable:

- Como una lista de elementos separados por comas (la lista puede estar vacía)
- Como una tabla Ambas técnicas se muestran en el siguiente código.

El compilador trata las dos técnicas de la misma forma.

```
static void Main( )
{
  long x;
  x = Sumador (63, 21, 84); // Lista
  x = Sumador (new long[ ]{ 63, 21, 84 }); // Tabla
}
```

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 recursivos

- Es un método que puede hacer una llamada a sí mismo.
  - Directamente.
  - Indirectamente.

- Útil para resolver ciertos problemas
  - Árboles.
  - Listas.

Recursividad

- Poderosa herramienta de programación como alternativa a algoritmos iterativos.
- Soluciones elegantes a problemas difíciles de resolver de otro modo (forma iterativa)
- **Un método es recursivo si contiene invocaciones a sí mismo**
- Una llamada a un método recursivo puede generar una o más invocaciones al mismo método, que a su vez genera otras, …

Módulos en C#• FIUAS

Cuando utilizar Recursividad

- En general, las soluciones recursivas son menos eficientes que las iterativas (costo mayor en tiempo y memoria).
- Consejos:
  - Los algoritmos que por naturaleza son recursivos y donde la solución iterativa es complicada y debe manejarse explícitamente una pila para emular las llamadas recursivas, deben resolverse por métodos recursivos.
  - Cuando haya una solución obvia al problema por iteración, debe evitarse la recursividad.

Plantilla solución de Izq-Der

```
TipoDato NombreMetodo (parametros)
{
    //definición  de variables locales

    //declaración del caso base

    //Trabajo

    //Llamada recursiva
}
```

Módulos en C#• FIUAS

Plantilla solución de Der - Izq

```
TipoDato NombreMetodo (parametros)
{
    //definición  de variables locales

    //declaración del caso base

    //Llamada recursiva

    //Trabajo
}
```

Codificación de Algoritmos al Lenguaje.

- Elabore un algoritmo que solicite al usuario un número entero, calcule la suma de los números secuenciales desde cero hasta ese número.

Uso de métodos sobrecargados

Los métodos no pueden tener el mismo nombre que otros elementos en una clase.

Sin embargo, dos o más métodos en una clase sí pueden compartir el mismo nombre. A esto se le da el nombre de sobrecarga.

- 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));
    }
}
```

Módulos en C#• FIUAS

Declaración de métodos sobrecargados

Los métodos sobrecargados son métodos que tienen el mismo nombre dentro de una clase.

El compilador de C# distingue métodos sobrecargados comparando las listas de parámetros.

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

Módulos en C#• FIUAS

Uso de métodos sobrecargados

- Conviene usar métodos sobrecargados si:
  - Hay métodos similares que requieren parámetros diferentes.
  - Se quiere añadir funcionalidad al código existente.

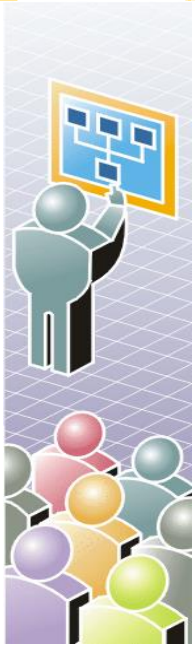
- No hay que abusar, ya que:
  - Son difíciles de depurar.
  - Son difíciles de mantener.

Codificación de Algoritmos al Lenguaje.

- Elabore un algoritmo que pueda sumar dos números reales, enteros o cadenas según lo indique el usuario(métodos sobrecargados), emplee métodos y el tipo de parámetro apropiado.

Actividad #1 Extraclase

- Descripción:
  - Codificar el siguiente pseudocódigo que calcula e imprime:
    - El área de un triangulo (Base * Altura /2), y



Propuesta Algoritmo Modular

```
//Objetivo: Calcular el área de un Triangulo
//Programador: MC. Gálvez
//Fecha: ___de Septiembre de 2015
INICIO
   //Definición de Constantes  y  Variables Globales

   PRINCIPAL ()
   INICIO
         //Definición de Constantes  y  Variables Locales
         REAL Area, Base, Altura
         LecturaDatos(Altura,Base)
         Area= CalcularArea(Altura, Base)
         ImprimirArea(Area)
   FIN

   SINVALOR  LecturaDatosTriangulo(REAL Altura, REAL Base)
   INICIO
        IMPRIMIR "Proporciona el valor de la Base: "
        LEER Base
        IMPRIMIR "Proporciona el valor de la Altura: "
        LEER Altura
        IMPRIMIR "Fin de lectura…"
   FIN
```


Continuación …

```
   REAL  CalcularArea(REAL Altura, REAL Base)
   INICIO
        //Definición de Constantes  y  Variables Locales
        REAL Area
        ///Proceso, calcular el área
        Area=Base * Altura / 2
        //Regresar el resultado obtenido
         REGRESAR Area
   FIN

   SINVALOR  ImprimirArea(REAL Area)
   INICIO
         IMPRIMIR "El Área es: ", Area
   FIN

FIN
```

Preguntas?