Repaso de declaración y ámbito o alcance de variables

Declaración de variables

Recordamos la sintaxis básica para declarar variables:

[Ámbito] [Tipo Dato] [Nombre Variable] = [Dato asignado];

Ejemplos:

```
private int numero = 0;
public String texto;
Boolean bandera;
```

Ámbito o Alcance

Primero que todo cuando nos referimos a ámbito o alcance de una variable, nos referimos a la visibilidad que tendrá esta variable con respecto al proyecto que estamos creando. C# cuenta con 5 ámbitos distintos:

- **public**: Los elementos declarados serán accesibles desde cualquier porción de código del proyecto y desde cualquier otro proyecto que haga referencia a aquel donde están declarados. No se pueden utilizar dentro en el interior de las funciones.
- **protected**: Se puede utilizar en el interior de una clase. Permite restringir el acceso a la variable únicamente al código de la clase y todas las clases que hereden de ella.
- **internal**: Serán accesibles desde el ensamblado en el cual están declarados y tampoco se pueden utilizar en el interior de una función.
- **protected internal**: Es el nivel de acceso de protected e internal, juntos.
- **private**: restringe el acceso a la variable al módulo, a la clase o a la estructura en la cual está declarada. No se puede utilizar en el interior de un procedimiento o función.

Si no se indica nada a la variable se considerará como **private**.

Tipo de datos VAR

A las variables locales se les puede asignar un "tipo" deducido en lugar de un tipo explícito, a esto se lo llama “tipado implícito”. Se logra con la palabra reservada var indica al compilador que deduzca el tipo de dato de la variable a partir de la expresión que se encuentra en el lado derecho de la instrucción de inicialización. El tipo deducido puede ser un tipo integrado, un tipo anónimo, un tipo definido por el usuario o un tipo definido en la biblioteca de clases de .NET Framework.

Ejemplo:

```
// i es compilado como un int
        var i = 5;

// s se compila como string
        var s = "Hola";

 // a se compila como int[]
        var a = new[] { 0, 1, 2 };

// expr se compila como IEnumerable<Cliente> o quizás como IQueryable< Cliente >
        var expr = from c in clientes
        where c.Ciudad == "Rosario”
        select c;

// anon se compila como un tipo anónimo
        var anon = new { Name = "Juan", Age = 34 };

// list se compila como una List<int>
        var list = new List<int>();
```

Repaso de estructuras de control

Recordamos las estructuras de control para formar el flujo de código de un programa:

Estructura IF

```
int a = 5;
int b = 6;
if (a + b > 10)
        Console.WriteLine("La suma de ambas variables es mayor a 10.);
```

Estructura IF ELSE

```
int a = 5;
int  b = 3;
if (a + b > 10)
        Console.WriteLine("La suma de ambas variables es mayor a 10");
else
        Console.WriteLine("La suma de ambas variables es menor o igual a 10.");
```

La siguiente solución en resultado es equivalente a la anterior. Se agregan las { } (llaves) delimitadoras de bloque de instrucciones. Con la sintaxis anterior, no se pueden agregar varias instrucciones para el if o varias instrucciones para el else.

```
int a = 5; int
b = 3;
if (a + b > 10)
        {
        Console.WriteLine("La suma de ambas variables es mayor a 10");
        }
else
        {
        Console.WriteLine("La suma de ambas variables es menor o igual a 10.");
        }
```

Estructura SWITCH

```
int b = 3;
switch (b)
```

```
{
        case 1:  Console.WriteLine("El valor es 1");

        break;

        case 2:  Console.WriteLine("El valor es 2");

        break;

        case 3:  Console.WriteLine("El valor es 3");

        break;
        default: Console.WriteLine("El valor no es 1 ni 2 ni 3");

        break;

}
```

Estructura WHILE

```
int contador = 0;
while (contador < 10) {
        Console.WriteLine($"Hola! El contador tiene {contador}");
        contador ++;
}
```

Estructura DO WHILE

```
int contador = 0;
do
{
        Console.WriteLine($"Hola! El contador tiene {contador}");
        contador ++;
} while  (contador < 10);
```

Estructura FOR

```
for(int contador = 0; contador < 10; contador ++)
{
        Console.WriteLine($"Hola! El contador tiene {contador}") ;
}
```

SWITCH vs IF Else

Una instrucción switch se traduce más fácilmente a una tabla de salto (o rama). Esto puede hacer que las declaraciones de conmutación sean mucho más eficientes que if-else cuando las etiquetas de caso están muy juntas. La idea es colocar un montón de instrucciones de salto secuencialmente en la memoria y luego agregar el valor al contador del programa. Esto reemplaza una secuencia de instrucciones de comparación con una operación add.

A continuación se presentan algunos ejemplos de pseudo-ensamblaje extremadamente simplificados. Nota, solo se muestra el presente ejemplo para que comprenda la diferencia entre estos comandos. Primero, la versión if-else:

// Código C#

```
if (1 == value)
        function1();
else if (2 == value)
        function2();
else if (3 == value)
        function3();
```

// Versión del ensamblado del código anterior

```
compare value, 1 jump if zero label1
compare value, 2 jump if zero label2
compare value, 3 jump if zero label3
label1: call function1
label2: call function2
label3: call function3
```

Observe ahora la versión SWITCH

// Código C#

```
switch (value) {
case 1: function1(); break;
case 2: function2(); break;
case 3: function3(); break;
}
```

// Versión ensamblada

```
add program_counter, value
call function1 call function2
call function3
```

Resumen: Utilice IF Else sólo cuando el objeto a validar sea diferente en cada caso. En el resto de los casos utilice siempre un bloque Switch. Recuerde colocar Break en caso de que necesite el corte del caso, en cambio cuando un caso ejecute el mismo código que otro colóquelos juntos sin un Break entre ellos.

Para repasar programación básica se sugiere consultar el siguiente link Estructuras de control.