

Estructuras y enumeraciones

En este capítulo aprenderemos un tema muy útil. Empezaremos por cómo utilizar los tipos definidos por el programador, es decir, los tipos de datos definidos por nosotros. Estos tipos nos darán la flexibilidad de poder guardar lo que nosotros necesitemos en nuestra aplicación y poder utilizarlos de forma similar a las variables.

Las estructuras	280
Cómo definir una estructura	280
Cómo crear una variable del nuevo tipo	282
Cómo acceder a los campos de la estructura	283
Cómo mostrar los campos de la estructura	283
Creación de un constructor para la estructura	286
Estructuras enlazadas	299
Las enumeraciones	309
Resumen	315
Actividades	316

Las estructuras

Las estructuras son tipos definidos por el programador y son un conjunto de datos agrupados. Supongamos que programamos una agenda telefónica. Desde luego, la agenda necesita guardar mucha información, por lo que ya hemos pensado en usar arreglos. Pero también necesitamos pensar en los datos que debe contener como: nombre, edad, teléfono. Esto nos lleva a tener tres arreglos. Si necesitamos más datos, entonces más arreglos serán necesarios. Al final es posible que sea un poco complicado administrar tantos arreglos y esto reduce la flexibilidad de nuestro software. Lo mejor sería poder agrupar esa información, de forma tal que solamente tengamos que administrar un solo arreglo. La forma de poder agrupar la información es por medio del uso de las estructuras.

Cuando creamos una estructura, definimos un nuevo tipo y adentro de este tipo podremos colocar datos. Estos datos se conocen como **campos**. Cada campo está representado por una variable, que puede ser de cualquier tipo. Luego que hemos terminado de definir la estructura podemos crear variables de esta estructura y guardar la información que necesitemos en ellas.

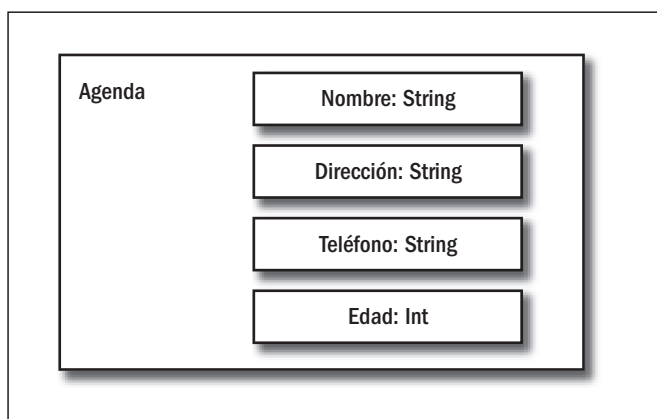


Figura 1. Ésta es una representación de una estructura y sus campos.

Cómo definir una estructura

Definir una estructura es sencillo. Sin embargo, el primer paso no se lleva a cabo en la computadora. Lo primero que tenemos que hacer es encontrar cuáles son los campos y el tipo que debe guardar la estructura. En el programa la definiremos utilizando el código que comentamos en el siguiente bloque:

```
acceso struct nombre
{
```

```

    acceso tipo campo1;
    ...
    acceso tipo campoN;

}

```

El **acceso** indica si la estructura puede verse por afuera del ámbito donde ha sido definida o no. Si el acceso es de tipo **público**, se puede acceder a la estructura por afuera del ámbito. Para esto pondremos como acceso **public**. Si no deseamos que el exterior pueda acceder a la estructura, entonces la colocamos como **privada**. Para esto colocamos el acceso como **private**. En este libro siempre trabajaremos con acceso de tipo **public**. Para indicar que definimos una estructura usamos **struct** seguido del nombre de la estructura. Éste puede ser cualquier nombre válido en C#. No hay que confundir el nombre de la estructura con la o las variables que usaremos de ella. Con este nombre identificaremos el nuevo tipo.

Luego tenemos que crear un bloque de código. Dentro de este bloque definiremos los campos que necesitamos. Éstos se definen igual que las variables, pero es necesario colocar el acceso. En este libro utilizaremos un acceso de tipo **public**, de forma tal que podamos leer la información contenida adentro de la estructura.

Por ejemplo, la estructura de nuestra agenda puede ser definida de la siguiente forma:

```

public struct Agenda
{
    public String Nombre;
    public int Edad;
    public String Telefono;
}

```

Así de sencilla es la definición. Ya tenemos una estructura llamada **Agenda** que contiene los campos: **Nombre**, **Edad** y **Telefono**.

III

SELECCIÓN ADECUADA DE LOS CAMPOS

El hacer un análisis correcto de los campos y los tipos que debe tener una estructura nos será de mucha ayuda. Esto nos ahorra cambios que pueden ser muy costosos en tiempo y mantenimiento de nuestro programa. La lista de campos surge de manera similar a la lista de variables que hemos aprendido en el **Capítulo 2**.

Nosotros podemos definir tantas estructuras como sean necesarias y también cada estructura puede tener la cantidad necesaria de campos.

Cómo crear una variable del nuevo tipo

Ya que tenemos definida la estructura, es necesario poder definir sus variables para guardar información. Como hemos realizado la declaración de un nuevo tipo, entonces nos será posible crear variables de ésta, y la forma de hacer esto es similar a la de cualquier otro tipo.

Para definir la variable sólo deberemos poner el nombre del tipo seguido del nombre que le daremos a nuestra variable. No debemos olvidar colocar ; (punto y coma) al final de la sentencia. Supongamos que deseamos crear una variable cuyo nombre será **amigo** y es del tipo agenda.

```
Agenda amigo;
```

Ahora, adentro del nuevo tipo de variable declarada llamada **amigo**, se encuentran los campos **Nombre**, **Edad** y **Telefono**. Es posible declarar varias variables de **Agenda**.

```
Agenda amigo1, amigo2, amigo3;
```

En el caso de necesitar hacer uso de éste, podemos crear un arreglo de la estructura. Este arreglo puede ser del tamaño que necesitemos para nuestra aplicación, y es entonces de esta forma que para la estructura agenda podemos tener un arreglo donde guardamos la información de los amigos, y otro arreglo donde se guardará la información de los clientes.

```
Agenda []amigos=new Agenda[15];  
Agenda []clientes=new Agenda[5];
```



CREACIÓN DE VARIABLES DE ESTRUCTURA

Podemos definir una variable de la estructura de la forma tradicional, pero también podemos hacerlo por medio **new**. Con ésta última es posible colocar un constructor adentro de la estructura y facilitarnos la introducción de información. Si se define de forma tradicional deberemos inicializar cada campo de la estructura manualmente.

Cómo acceder a los campos de la estructura

Ya tenemos variables de la estructura y sabemos que adentro de éstas se encuentran los campos que guardarán la información. Tenemos que tener acceso a los campos para poder guardar, leer o modificar los datos. El acceso al campo se lleva a cabo de la siguiente manera:

```
VariableEstructura.Campo
```

Esto quiere decir que primero debemos indicar cuál es la variable con la que deseamos trabajar. Luego necesitamos utilizar el operador **punto** y seguido de él colocamos el nombre del campo a acceder. Veamos un ejemplo de esto:

```
amigo.Edad = 25;
```

En este caso decimos que al campo **Edad** de la variable **amigo** le colocamos en su interior el valor **25**. Podemos utilizar los campos en cualquier tipo de expresión válida, tal y como una variable:

```
if(amigo.Edad>18)
...
diasVividos=amigo.Edad * 365;
...
Console.WriteLine("El nombre es {0}",amigo.Nombre);
```

Cómo mostrar los campos de la estructura

Algo que necesitaremos constantemente es presentar los campos de la estructura. La manera más evidente de hacerlo es con el método **WriteLine()**. Simplemente mostramos el contenido del campo como si fuera otra variable del programa.



CAMPOS OLVIDADOS

Es posible que tengamos una estructura y que sólo utilicemos determinados campos, sin embargo, los demás campos continúan existiendo. Algunos de éstos pueden inicializarse automáticamente a **0** o **null** si usamos **new** para crear la variable. Pero si se hace de forma tradicional y no se han inicializado, entonces ese campo no tiene instancia y originará problemas si se intenta usar.

```
Console.WriteLine("La edad es {0}", amigo.Edad);
```

Esta forma es sencilla y útil en los casos en los que solamente necesitamos mostrar un campo de la estructura. Sin embargo, de necesitar mostrar todos los campos, deberemos colocar **WriteLine()** para cada campo, o uno solo con una cadena de formato larga. En cualquier caso resulta incómodo.

La mejor forma de mostrar todos los datos contenidos en los campos sería mediante la conversión de la estructura a una cadena. Sabemos que existe un método llamado **ToString()** que hemos usado con las variables numéricas. Sin embargo, C# no se lo puede dar directamente a nuestra estructura porque no puede saber cuáles son los campos que contiene ni cómo los deseamos mostrar. Por esto, cae en nuestra responsabilidad programar el método **ToString()** de nuestra estructura.

Veamos esto en el siguiente ejemplo:

```
public struct Agenda
{
    public String Nombre;
    public int Edad;
    public String Telefono;

    public override String ToString()
    {
        StringBuilder sb = new StringBuilder();
        sb.AppendFormat("Nombre: {0}, Edad: {1},
            Telefono: {2}", Nombre, Edad,
            Telefono);
        return (sb.ToString());
    }
}
```

Podemos observar que adentro del bloque de código de la estructura hemos colocado nuestra versión del método **ToString()**. El acceso debe ser público para que se pueda invocar desde el exterior de la estructura. Este método debe regresar un objeto de tipo **String** y no necesita ningún parámetro.

En el interior del método simplemente colocamos el código necesario para darle formato a la cadena y en nuestro caso hacemos uso de **StringBuilder**, aunque es válido usar cualquier otro método. Al finalizar ya tenemos la cadena con la información de nuestros campos y la regresamos por medio de **return**.

Ahora que ya tenemos implementado **ToString()** mostrar la información contenida será tan sencillo como:

```
Console.WriteLine(amigo.ToString());
```

Veamos un ejemplo completo de estos conceptos:

```
public struct Agenda
{
    public String Nombre;
    public int Edad;
    public String Telefono;

    public override String ToString()
    {
        StringBuilder sb = new StringBuilder();
        sb.AppendFormat("Nombre: {0}, Edad: {1},\n",
            Telefono: {2}", Nombre, Edad,
            Telefono);
        return (sb.ToString());
    }
}

static void Main(string[] args)
{
    //
    // TODO: agregar aquí código para iniciar la aplicación
    //

    Agenda []amigos=new Agenda[5];

    amigos[1].Edad=25;
    amigos[1].Nombre="Juan";
    amigos[1].Telefono="(555) 123-4567";

    Console.WriteLine(amigos[1].ToString());
}
```

Como parte del ejemplo creamos un arreglo de la estructura **Agenda** que se llama **amigos** y tiene 5 elementos, luego, para el elemento en el índice 1 colocamos los datos. Para finalizar simplemente imprimimos el elemento 1 del arreglo **amigos**, pero usamos el método **ToString()** programado por nosotros. Si compilamos y ejecutamos el programa obtendremos lo siguiente:

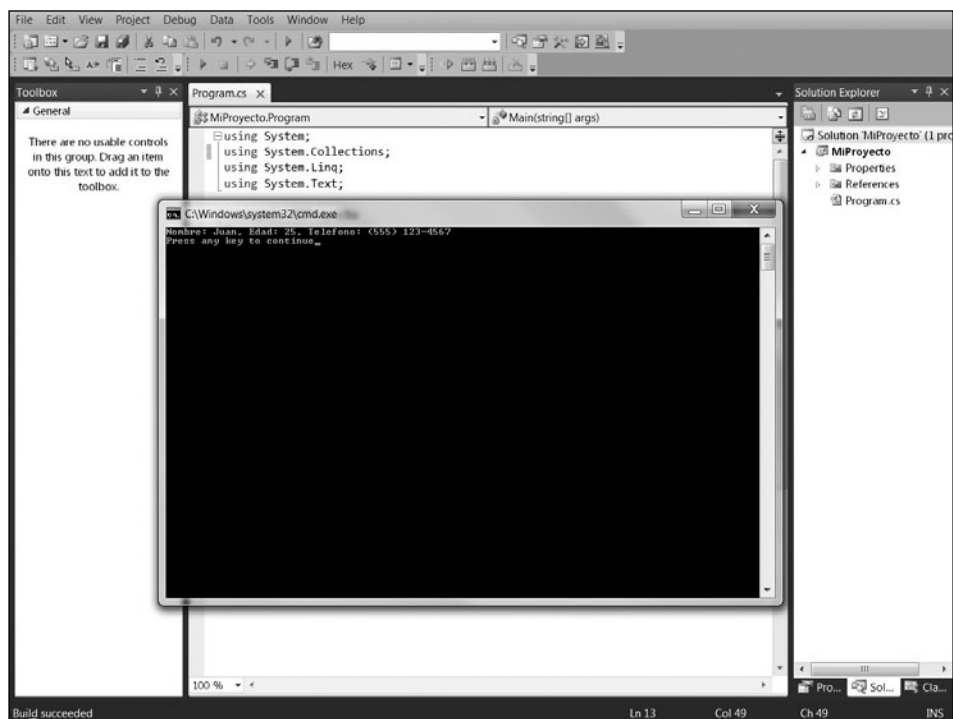


Figura 2. Podemos observar cómo por medio de **ToString()** obtenemos la cadena con la información de la estructura y la mostramos en la consola.

Creación de un constructor para la estructura

En el ejemplo anterior hemos visto la manera de cómo cada uno de los campos que componen la estructura ha tenido que inicializarse en forma individual. Esto es correcto y no presenta ningún problema, sin embargo, hay una forma de poder inicializar los campos más fácilmente, sin tantas complicaciones.

Para hacer esto podemos hacer uso de un **constructor**. El constructor no es otra cosa que un método que nos permitirá llevar a cabo la inicialización de los campos. Sin embargo, este método tiene algunas características especiales. La primera característica es que siempre se llamará igual que la estructura a la que pertenece. La segunda es muy importante: el constructor se invoca automáticamente cuando llevamos a cabo la instanciación de la variable de la estructura. La última característica del constructor es que no tiene tipo. No sólo no regresa nada, no tiene tipo.

Adentro del constructor podemos colocar cualquier código válido de C#, pero es evidente que colocaremos código dedicado a la inicialización de los campos. Veamos un primer ejemplo de cómo podemos crear un constructor. El constructor siempre va adentro del bloque de código de la estructura.

```
public Agenda(String pNombre, int pEdad, String
    pTelefono)
{
    // Llevamos a cabo la asignación
    Nombre=pNombre;
    Edad=pEdad;
    Telefono=pTelefono;
}
```

En el código del constructor vemos que el acceso es público. Esto es necesario y siempre debemos dejarlo así. Si observamos luego se coloca directamente el nombre del constructor. El nombre es **Agenda**, ya que pertenece a la estructura **Agenda**. A continuación tenemos la lista de parámetros. Los valores pasados como parámetros serán asignados a los campos correspondientes. En la declaración de la variable lo tendremos que usar de la siguiente forma:

```
Agenda amigo=new Agenda("Juan",25,"(555) 123-4567");
```

Aquí vemos que declaramos la variable **amigo** que es de tipo **Agenda**. Al hacer la instanciación por medio de **new** vemos que pasamos los parámetros. Éstos serán los datos que quedarán guardados en los campos de la variable **amigo**. La cadena "**Juan**" quedaría adentro del campo **Nombre**, el valor de **25** adentro del campo **Edad** y la cadena "**(555) 123-4567**" en el campo **Telefono**.

Veamos cómo quedaría un programa completo con el constructor:

{ } EL USO DE OVERRIDE

El método `ToString()` se definió en la clase `Object`, de la cual descende todo en C#, pero al estar definiendo una versión nueva para nuestra estructura usamos `override`, que le indica al compilador que creamos una nueva implementación que usará nuestra estructura. Así, al invocar `ToString()` se usará la versión implementada por nosotros en lugar de la versión base.

```

public struct Agenda
{
    public String Nombre;
    public int Edad;
    public String Telefono;

    public Agenda(String pNombre, int pEdad, String
        pTelefono)
    {
        // Llevamos a cabo la asignación
        Nombre=pNombre;
        Edad=pEdad;
        Telefono=pTelefono;
    }

    public override String ToString()
    {
        StringBuilder sb = new StringBuilder();
        sb.AppendFormat("Nombre: {0}, Edad: {1},
            Telefono: {2}", Nombre, Edad,
            Telefono);
        return (sb.ToString());
    }
}

static void Main(string[] args)
{
    //
    // TODO: agregar aquí código para iniciar la aplicación
    //
}

```



TUTORÍA DE ESTRUCTURAS

En la dirección web que listaremos a continuación, encontraremos una pequeña tutoría sobre la utilización de las estructuras en C#. Recordemos siempre buscar información, tutoriales y cualquier otro material que nos sea de utilidad, para poder ampliar nuestros conocimientos: www.devjoker.com/asp/ver_contenidos.aspx?co_contenido=161.

```

        Agenda amigo=new Agenda("Juan",25,"(555) 123-4567");

        Console.WriteLine(amigo.ToString());
    }

```

Ejecutemos el programa y veamos el resultado que obtenemos:

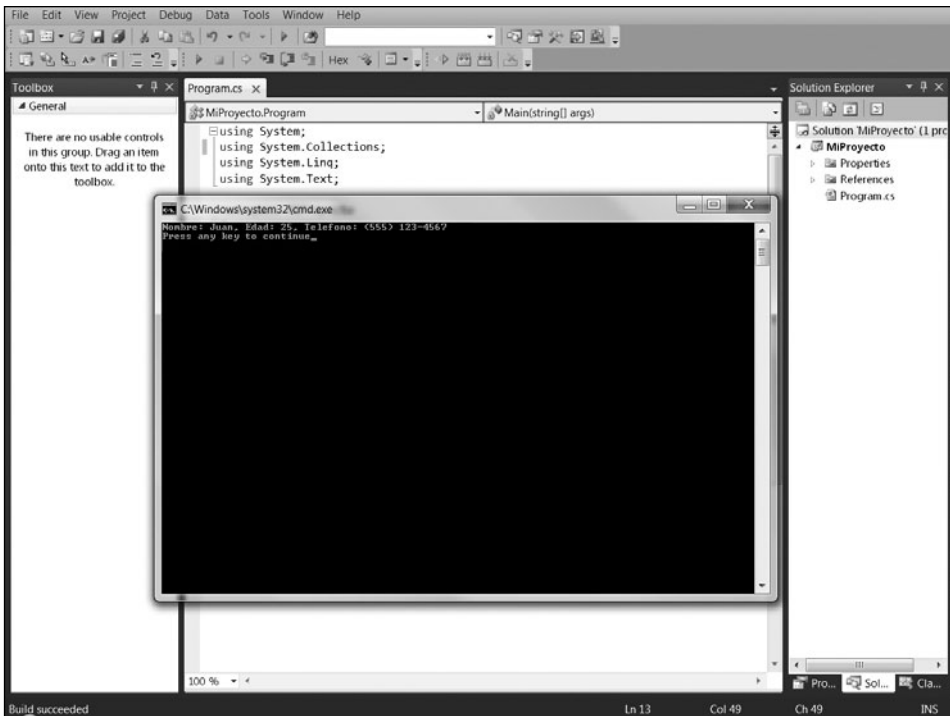


Figura 3. Vemos cómo efectivamente los datos han sido colocados en los campos de la variable por medio del constructor.

Cómo usar el constructor para validar información

El constructor no solamente puede ser utilizado para colocar la información adentro de los campos, una de las grandes ventajas que éste nos da es la posibilidad de poder validar la información antes de que sea asignada a uno de sus campos. Es posible que existan ciertas reglas sobre lo que se considera información válida para nuestros campos, y por supuesto, podemos implementar estas mismas en el constructor.

Supongamos que el teléfono sólo se considera válido si tiene más de 8 caracteres. En caso de que sea menor es inválido y no debe ser asignado para evitar problemas. Esta simple regla es fácil de programar en el constructor. En nuestro caso podría quedar de la siguiente forma:

```

public Agenda(String pNombre, int pEdad, String
               pTelefono)
{
    // Llevamos a cabo la asignación
    Nombre=pNombre;
    Edad=pEdad;

    if(pTelefono.Length>8)
        Telefono=pTelefono;
    else
        Telefono="Teléfono no válido";
}

```

Como vemos, hacemos uso de **if** y verificamos la longitud de la cadena **pTelefono**. Si es mayor a **8** entonces le asignamos **pTelefono** al campo **Telefono**. En caso contrario, colocamos un mensaje que dice que el teléfono no es válido.

El programa completo es el siguiente:

```

public struct Agenda
{
    public String Nombre;
    public int Edad;
    public String Telefono;

    public Agenda(String pNombre, int pEdad, String
                  pTelefono)
    {
        // Llevamos a cabo la asignación
        Nombre=pNombre;
    }
}

```



TUTORÍA DE ENUMERACIÓN

El mismo sitio web que nombramos anteriormente presenta una tutoría sobre el uso de la enumeración en C# que complementa los conceptos que hemos aprendido en este capítulo:

www.devjoker.com/contenidos/Tutorial-C/164/Enumeraciones.aspx.

```

        Edad=pEdad;

        if(pTelefono.Length>8)
            Telefono=pTelefono;
        else
            Telefono="Telefono no valido";
    }

    public override String ToString()
    {
        StringBuilder sb = new StringBuilder();
        sb.AppendFormat("Nombre: {0}, Edad: {1},
            Telefono: {2}", Nombre, Edad,
            Telefono);
        return (sb.ToString());
    }

}

static void Main(string[] args)
{
    //
    // TODO: agregar aquí código para iniciar la aplicación
    //

    Agenda amigo=new Agenda("Juan",25,"(555) 123-4567");
    Agenda amigo1=new Agenda("Pedro",32,"(555)");
    Console.WriteLine(amigo.ToString());
    Console.WriteLine(amigo1.ToString());
}

```



TUTORÍA SOBRE EL CONSTRUCTOR EN LAS ESTRUCTURAS

Si deseamos ampliar los conceptos y conocimientos sobre el uso del constructor en las estructuras, podemos visitar las tutorías del sitio web Devjoker: www.devjoker.com/contenidos/Tutorial-C/163/Constructores-de-estructuras.aspx.

```
}
```

Si compilamos y ejecutamos el programa tendremos el siguiente resultado:

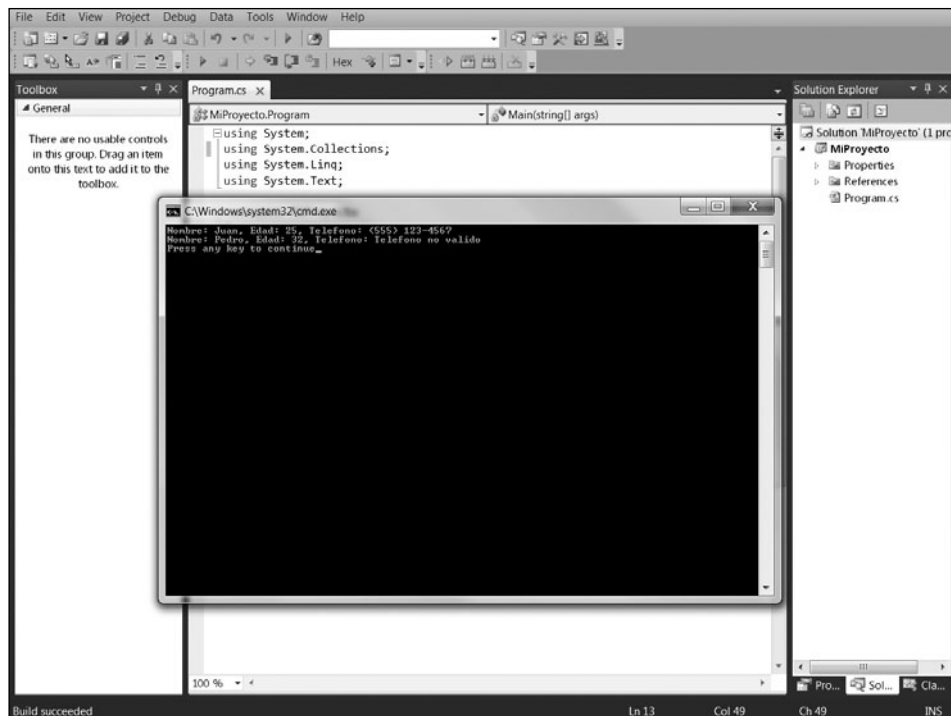


Figura 4. Podemos ver cómo el constructor no sólo ha asignado el valor sino que también ha validado la información.

La sobrecarga del constructor

El constructor puede ser sobrecargado y esta habilidad nos brinda mucha flexibilidad para poder utilizarlo. Supongamos que tenemos amigos que tienen teléfono y otros que no. Si no tienen teléfono, no tiene sentido utilizar un constructor que lo solicite. Lo mejor sería tener una segunda versión del constructor que solamente

III LA SOBRECARGA

Sobrecarga es una técnica de programación que permite tener varias versiones de una función o un método. El compilador selecciona la versión a utilizar basándose en la cantidad de parámetros y los tipos. Esto nos permite agrupar métodos diferentes bajo un mismo nombre o concepto, facilitando la programación, solamente recordando un nombre, y no los nombres para cada versión.

necesite el nombre y la edad, y podemos hacer que esta versión coloque un mensaje que indique que no tiene teléfono.

Para hacer la sobrecarga colocamos la segunda versión debajo de la que ya tenemos.

Por ejemplo, nuestra segunda versión queda de la siguiente forma:

```
public Agenda(String pNombre, int pEdad)
{
    // Llevamos a cabo la asignación
    Nombre=pNombre;
    Edad=pEdad;

    Telefono="Sin teléfono";
}
```

Como podemos ver, solamente recibimos dos parámetros, pero aun así los tres campos son asignados con un valor. Ningún campo se queda sin asignar. Para declarar la variable le pasamos los 2 parámetros necesarios.

```
Agenda amigo2=new Agenda("Luis",28);
```

Veamos un ejemplo que utilice la sobrecarga del constructor de la estructura:

```
public struct Agenda
{
    public String Nombre;
    public int Edad;
    public String Telefono;

    public Agenda(String pNombre, int pEdad, String
        pTelefono)
    {
        // Llevamos a cabo la asignación
        Nombre=pNombre;
        Edad=pEdad;
        if(pTelefono.Length>8)
```

```

        Telefono=pTelefono;
    else
        Telefono="Teléfono no válido";
    }

    public Agenda(String pNombre, int pEdad)
    {
        // Llevamos a cabo la asignación
        Nombre=pNombre;
        Edad=pEdad;

        Telefono="Sin teléfono";
    }

    public override String ToString()
    {
        StringBuilder sb = new StringBuilder();
        sb.AppendFormat("Nombre: {0}, Edad: {1},
            Telefono: {2}", Nombre, Edad,
            Telefono);
        return (sb.ToString());
    }
}

static void Main(string[] args)
{
    //
    // TODO: agregar aquí código para iniciar la aplicación
    //

    Agenda amigo=new Agenda("Juan",25,"(555) 123-4567");
    Agenda amigo1=new Agenda("Pedro",32,"(555)");
    Agenda amigo2=new Agenda("Luis",28);

    Console.WriteLine(amigo.ToString());
    Console.WriteLine(amigo1.ToString());
    Console.WriteLine(amigo2.ToString());
}

```


Si compilamos la aplicación, veremos el resultado en la siguiente figura:

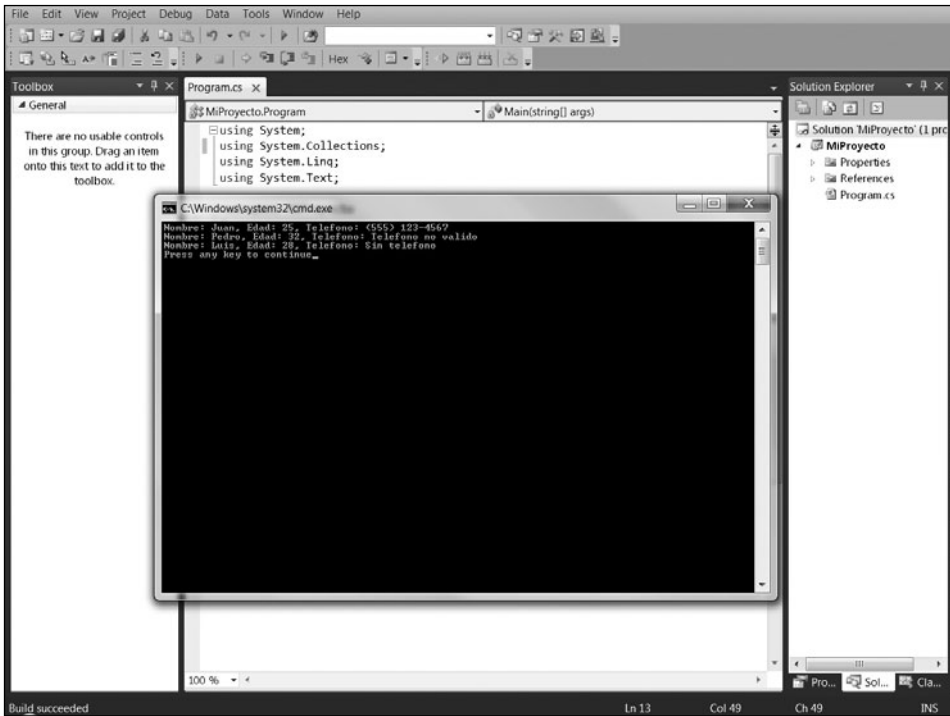


Figura 5. En esta figura podemos observar cómo el constructor sobrecargado también ha asignado la información necesaria en los campos.

Incluso podemos hacer que el constructor le solicite la información directamente al usuario. Hay que recordar que podemos colocar cualquier código válido en su interior. Esto lo logramos al crear una nueva sobrecarga.

Veamos cómo puede quedar esta nueva versión del constructor:

```
public Agenda(String pNombre)
{
    // Asignamos el nombre
    Nombre=pNombre;

    // Pedimos la edad
    Console.WriteLine("Dame la edad:");
    Edad=Convert.ToInt32(Console.ReadLine());
}
```

```

        // Pedimos el teléfono
        Console.WriteLine("Dame el teléfono:");
        Telefono=Console.ReadLine();

        if(Telefono.Length<8)
            Telefono="Sin teléfono";

    }

```

Lo primero que hacemos es asignar el nombre que hemos recibido por parámetro. Luego de la forma usual pedimos la edad, notando cómo está colocada la sentencia. Sabemos que **ToInt32()** necesita un parámetro que es la cadena a convertir y **ReadLine()** regresa una cadena. De esta forma, aprovechamos y colocamos todo el código adentro de una sola línea. Otra opción que podríamos haber utilizado es una función que pida el dato, tal y como lo vimos en un capítulo anterior. Enseguida solicitamos el teléfono y aprovechamos para validar la información. El programa completo queda de la siguiente manera:

```

public struct Agenda
{
    public String Nombre;
    public int Edad;
    public String Telefono;

    public Agenda(String pNombre, int pEdad, String
        pTelefono)
    {
        // Llevamos a cabo la asignación
        Nombre=pNombre;
        Edad=pEdad;
    }
}

```



CONSTRUCTOR EN LA ESTRUCTURA

Cuando utilizamos el constructor con nuestra estructura o cuando tenemos sobrecargas, tenemos que tener cuidado con la cantidad de parámetros. Dentro de las estructuras el constructor necesita tener al menos un parámetro ya que no es válido colocar un constructor sin parámetros. Si olvidamos colocar los parámetros, tendremos un error durante la compilación del programa.