

Programación orientada a objetos. Formularios Web

Indice

Nº 5. Programación orientada a objetos. Formularios Web.....	1
1. Tipos, objetos y espacios de nombres.....	1
De la programación estructurada al enfoque orientado a objetos.....	1
2. Elementos básicos de la POO.....	2
2.1. Las clases.....	2
2.2 Los Objetos.....	2
2.3 Los miembros de una clase.....	2
3. Los tres pilares de la Programación Orientada a Objetos.....	3
3.1 Herencia.....	3
3.2 Encapsulación.....	5
3.3 Polimorfismo.....	5
4. Jerarquía de clases.....	6
4.1 Relaciones entre objetos.....	6
4.2 Herencia.....	6
4.3 Pertenencia.....	7
4.4 Utilización.....	7
4.5 Reutilización.....	7
5. Caso práctico.....	7
5.1 Crear o definir una clase.....	7
5.2 Ejemplo de cómo usar la herencia.....	9
Ejemplo.....	12
5.3 Sobrecargar el constructor de las clases.....	15
5.4 Los campos y las propiedades.....	16
5.5 ¿Cómo declarar una propiedad como un procedimiento Property?.....	20
5.6 Propiedades de sólo lectura.....	21
5.7 Propiedades de sólo escritura.....	22
5.8 Campos de sólo lectura.....	22
5.9 Pasos por valor y referencia.....	25
6. Espacios de nombres (namespace) y ensamblados (assemblies).....	28
6.1 Espacios de nombres (namespace).....	28
6.2 Utilizar namespace.....	30
6.3 Importar namespace.....	30
6.4 Ensamblados (Assemblies).....	31
7. Desarrollar aplicaciones en ASP.NET.....	31
7.1 El entorno de desarrollo integrado.....	31
7.2 El explorador de soluciones.....	33
7.3 Añadir formularios Web.....	34
7.4 Diseñar una página web.....	36
8. Formularios Web.....	39
8.1 La Directiva de página.....	42
8.2 El tipo de documento.....	42
8.3 Página web completa.....	43
9. Escribir código.....	43
9.1 Controladores de eventos.....	44
9.2 Intellisense y Outlinig.....	47
9.3 Depuración.....	51
9.4 El servidor Web.....	53
9.5 Una depuración sencilla.....	54

Indice

Ejercicios.....	58
Ejercicio 1.....	58
Ejercicio 2.....	59

Nº 5. Programación orientada a objetos.

Formularios Web

1. Tipos, objetos y espacios de nombres

Sigamos con los últimos conceptos de teoría antes de meternos de lleno con nuestro entorno de desarrollo integrado y empezar a hacer páginas asp.net mas serias...

Como hemos comentado en muchas ocasiones todo .NET Framework está basado en clases (u objetos). A diferencia de las versiones anteriores de Visual Basic, la versión .NET de este lenguaje basa su funcionamiento exclusivamente en las clases contenidas en .NET Framework, además sin ningún tipo de limitaciones. Debido a esta dependencia en las clases del .NET Framework y sobre todo a la forma "hereditaria" de usarlas, Visual Basic .NET tenía que ofrecer esta característica sin ningún tipo de restricciones. Así que nos toca aprender o coger una pequeña base de la programación orientada a objetos (POO en adelante). No es más fácil ni mas difícil simplemente es distinta. Una vez aprendida sólo utilizaremos este tipo de programación en lugar de la utilizada en lenguajes anteriores.

La POO es una evolución de la programación por procedimientos llamada también estructurada. Se basaba en funciones y procedimientos y el código que controlaba el flujo de las llamadas a estos. En ASP y Visual Basic, sobre todo en versiones anteriores se sigue programando mucho así. A veces por desconocimiento y otras por "miedo" no se da el salto a la POO. Los que vengan de versiones anteriores de ASP y Visual Basic me habéis entendido muy bien que significa lo de miedo. Y es que un programador con experiencia en ASP y VB puede hacer magníficas aplicaciones sin utilizar la POO (y sin aprovecharse de sus ventajas) , y sobre todo, en un tiempo relativamente reducido. Si ahora este programador debe aprender otra forma de programar e instrucciones, lógicamente produce un poco de respeto porque "ahora estoy muy cómodo". Sin embargo, y por propia experiencia, hay que hacer el pequeño esfuerzo de "cambiar de chip" mas que nada porque .NET tiene cuerda para rato y su filosofía y ventajas están en la POO.

De la programación estructurada al enfoque orientado a objetos

En la programación estructurada, (con nuestros procedimientos y funciones) el crecimiento de una aplicación hace que el mantenimiento de esta aplicación sea una laboriosa tarea debido a la gran cantidad de procedimientos y funciones que están interrelacionadas entre sí con multitudes de llamadas. Si necesitamos hacer una pequeña modificación nos puede suponer el realizar un repaso completo a ver a que funciones puede afectar este cambio... El código se encuentra disperso a lo largo del programa por lo que el mantenimiento se hará progresivamente mas difícil.

La organización de una aplicación en POO se realiza mediante estructuras de código que contienen un conjunto de procedimientos e información que ejecutan una serie de procesos destinados a resolver un grupo de tareas. Una aplicación orientada a objetos tendrá tantas estructuras de código como aspectos del programa tengamos que resolver.

Un procedimiento situado dentro de una estructura no podrá llamar si ser llamado por otro de una estructura distinta sino es bajo una serie de reglas. Lo mismo pasa con los datos, estarán separados del exterior excepto los que designemos. Este concepto de estructura es lo que llamaremos "Objeto"

Así que estos objetos tienen una información precisa y un comportamiento detallado realizado por procedimientos que hacen puedan clasificarse según el tipo de tarea que realizan.

Veamos ahora los conceptos básicos de la POO...

2. Elementos básicos de la POO

2.1. Las clases

Ya hemos hablado en varias ocasiones de las clases y hemos comentado que todo lo que tiene .NET Framework son clases. Una clase no es ni más ni menos que código. Aunque dicho de esta forma, cualquier programa sería una clase.

Cuando definimos una clase, realmente estamos definiendo dos cosas diferentes: los datos que dicha clase puede manipular o contener y la forma de acceder a esos datos.

Por ejemplo, si tenemos una clase de tipo Cliente, por un lado tendremos los datos de dicho cliente y por otro la forma de acceder o modificar esos datos. En el primer caso, los datos del Cliente, como por ejemplo el nombre, domicilio etc., estarán representados por una serie de campos o propiedades, mientras que la forma de modificar o acceder a esa información del Cliente se hará por medio de métodos. Esas propiedades o características y las acciones a realizar son las que definen a una clase.

Veamos de otra forma: una clase es el conjunto de especificaciones o normas que definen cómo se va a crear un objeto, es decir, las instrucciones para crear ese objeto. Una clase es la representación abstracta de algo y el objeto es la representación concreta de lo que una clase define. Tranquilo y sigue leyendo...

2.2 Los Objetos

Por un lado tenemos una clase que es la que define un "algo" con lo que podemos trabajar. Pero para que ese "algo" no sea un "nada", tendremos que poder convertirlo en "algo tangible", es decir, tendremos que tener la posibilidad de que exista. Aquí es cuando entran en juego los objetos, ya que un objeto es una clase que tiene información real.

Digamos que la clase es la "plantilla" a partir de la cual podemos crear un objeto en la memoria. Por ejemplo, podemos tener varios objetos del tipo Cliente, uno por cada cliente que tengamos en nuestra cartera de clientes, pero la clase sólo será una. Dicho de otra forma: podemos tener varias instancias en memoria de una clase. Una instancia es un objeto (los datos) creado a partir de una clase (la plantilla o el código).

En nuestros formularios: tenemos 10 botones y ... han sido creados a partir de la clase "Botón" (esto es contundente para que lo entiendas). La clase es la teoría y el objeto la práctica. El manual de instrucciones para crear el objeto. Y con el ejemplo del coche... podemos crear diferentes coches a partir de la clase "Coche" y cada uno puede tener sus propias propiedades: color, ... pero funcionan todos igual

2.3 Los miembros de una clase

Las clases contienen datos, esos datos suelen estar contenidos en variables. A esas variables cuando pertenecen a una clase, se les llama: campos o propiedades.

Por ejemplo, el nombre de un cliente sería una propiedad de la clase Cliente. Ese nombre lo almacenaremos en una variable de tipo String, de dicha variable podemos decir que es el "campo" de la clase que representa al nombre del cliente.

Programación orientada a objetos. Formularios Web

Por otro lado, si queremos mostrar el contenido de los campos que contiene la clase Cliente, usaremos un procedimiento que nos permita mostrarlos, ese procedimiento será un método de la clase Cliente.

Por tanto, los **miembros** de una clase son las propiedades (los datos) y los **métodos** las acciones a realizar con esos datos. (y esta es toda la complicación que tienen los miembros y los métodos)

Como te he comentado antes, el código que internamente usemos para almacenar esos datos o para, por ejemplo, mostrarlos, es algo que no debe preocuparnos mucho, simplemente sabemos que podemos almacenar esa información (en las propiedades de la clase) y que tenemos formas de acceder a ella, (mediante los métodos de dicha clase), eso es "abstracción" o encapsulación.

El proceso por el cual se obtiene un objeto a partir de las especificaciones de una clase se conoce como instanciación. Como hemos dicho el molde (la clase) puede crear los objetos con propiedades diferentes: varios botones en el formulario pero con distintos tamaños o colores (propiedades).

Verás en algunos ejemplo... y ahora instanciamos el objeto, esto quiere decir sin mas que "y ahora creamos el objeto"...

3. Los tres pilares de la Programación Orientada a Objetos

Todos los lenguajes basados en objetos debe cumplir estos tres requisitos:

1. **Herencia**
2. **Encapsulación**
3. **Polimorfismo**

Nota: Algunos autores añaden un cuarto requisito: **la abstracción**, pero este último está estrechamente ligado con la encapsulación, ya que, de echo, es prácticamente lo mismo, aunque ahora lo veremos...

3.1 Herencia

Esta es la característica mas importante de la POO, s Según la propia documentación de Visual Studio .NET:

Una relación de herencia es una relación en la que un tipo (el tipo derivado) se deriva de otro (el tipo base), de tal forma que el espacio de declaración del tipo derivado contiene implícitamente todos los miembros de tipo no constructor del tipo base.

Un poco mas claro.

La herencia es la capacidad de una clase de obtener la interfaz y comportamiento de una clase existente.

Y otra vez:

La herencia es la cualidad de crear clases que estén basadas en otras clases. La nueva clase heredará todas las propiedades y métodos de la clase de la que está derivada, además de poder modificar el comportamiento de los procedimientos que ha heredado, así como añadir otros nuevos.

Es decir, tengo una clase que es por ejemplo un coche, la herencia me dice que puedo crear un objeto coche a partir de este y que va a "heredar" todas sus características y además podré personalizarlo a mi gusto. (esto está mejor) Resumiendo: Gracias a la herencia podemos ampliar cualquier clase existente, además de aprovecharnos de todo lo que esa clase haga... De momento la clase es ese "coche" que os he comentado, enseguida veremos la definición de clase.

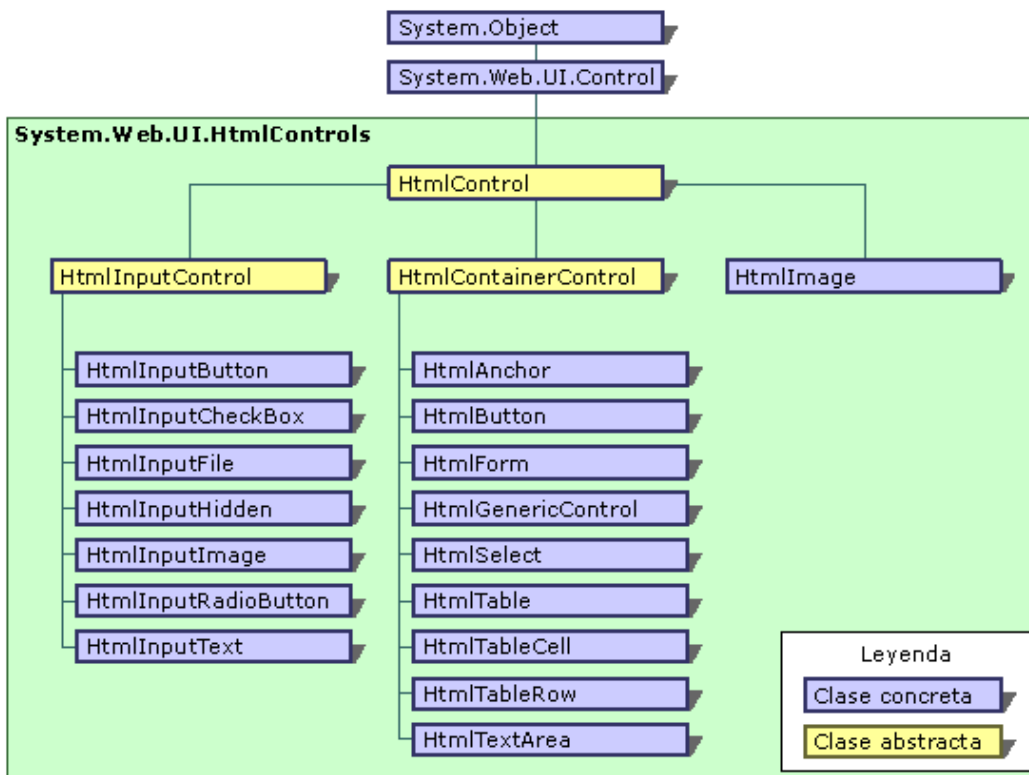
Para que lo entiendas mejor, veamos otro ejemplo *clásico*: Supongamos que tenemos una clase Gato que está derivada de la clase Animal. El Gato hereda de Animal todas las características comunes a los animales, además de añadirle algunas características particulares a su condición felina. Podemos decir que un Gato es un Animal, lo mismo que un Perro es un Animal, ambos están derivados (han heredado) de la clase Animal, pero cada uno de ellos es diferente, aunque en el *fondo* los dos son animales. Esto es

Programación orientada a objetos. Formularios Web

herencia: usar una clase base (Animal) y poder ampliarla sin perder nada de lo heredado, pudiendo ampliar la clase de la que se ha derivado (o heredado).

En ocasiones podrás ver los términos de otra forma, por ejemplo otra definición es que partiendo de una clase llamada clase base, padre o superclase (para decir jerárquicamente que es mas importante que la otra) podemos crear una nueva clase llamada derivada, hija o subclase. Así de una clase podemos derivar una subclase y otra dentro de esta... creando una jerarquía de clases.

Por ejemplo una jerarquía de clases sería como este gráfico:



Donde hay unos elementos de mas nivel que otros y que para referencia a los menores lógicamente tendré que indicar quién es su padre jerárquicamente.

Hay dos tipos de herencia:

- Simple. Cuando creamos una clase a partir de una sola clase base.
- Múltiple. Cuando creamos una clase a partir de varias clases base. Realmente complejo y no permitido por .NET así que tranquilo.

Repitiendo el ejemplo del coche, es una clase que tiene como todas las clases sus propiedades, métodos y eventos, por ejemplo.

- Propiedades: color del coche, modelo, tapicería, potencia... es decir, sus características
- Métodos: Acelerar, frenar, ...
- Eventos: está claro que está esperando muchos eventos, que se toque el claxon, la radio, una puerta... al dispararse un evento ejecutará una serie de acciones o un método.

Tenemos nuestra clase, vale pues podemos crear un "cochedeportivo" a partir de ella. Va a heredar todas sus características pero puedo modificar algunas cosas y añadirles otras por ejemplo un turbo o ABS.

3.2 Encapsulación

Otra vez, según Visual Studio.NET:

La encapsulación es la capacidad de contener y controlar el acceso a un grupo de elementos asociados. Las clases proporcionan una de las formas más comunes de encapsular elementos.

La encapsulación es la capacidad de separar la implementación de la interfaz de una clase del código que hace posible esa implementación. Esto realmente sería una especie de abstracción, ya que no nos importa cómo esté codificado el funcionamiento de una clase, lo único que nos debe interesar es cómo funciona...

Repetimos, tranquilo. Digamos que una clase es un elemento de Windows que como ya hemos visto tiene unas propiedades (color, fuente), métodos y eventos (clic), vale pues cuando decimos la implementación de la interfaz de una clase, nos referimos a los miembros de esa clase: métodos, propiedades, eventos, etc. Es decir, lo que la clase es capaz de hacer.

Cuando usamos las clases, éstas tienen una serie de características (los datos que manipula) así como una serie de comportamientos (las acciones a realizar con esos datos). La encapsulación es esa capacidad de la clase de ocultarnos sus interioridades para que sólo veamos lo que tenemos que ver, sin tener que preocuparnos de cómo está codificada para que haga lo que hace... simplemente nos debe importar que lo hace.

Si tomamos el ejemplo de la clase Gato, sabemos que araña, come, se mueve, etc., pero el cómo lo hace no es algo que deba preocuparnos, salvo que se lance sobre nosotros... aunque, en ese caso, lo que deberíamos tener es una clase "espanta-gatos" para quitárnoslo de encima lo antes posible... :-).

Ya ves, al final sólo es la capacidad de separar por un lado lo que nos muestra que hace ese objeto con el código interno que lo realiza. Así para nosotros un objeto por muy complejo que sea sólo veremos sus propiedades, métodos y eventos. Aplica esto a la clase botón, se que tiene esas propiedades, métodos y eventos pero no veo nada del código que realiza esas acciones, es decir, está **encapsulado**. Dicho finalmente de forma técnica: **Establece la separación entre la interfaz del objeto y su implementación.**

Esto aporta dos ventajas:

- Seguridad. Se evitan los accesos no deseados, es decir, si está bien encapsulada no podremos modificar directamente variables ni ejecutar métodos que sean de uso interno.
- Simplicidad. Un programador no debe conocer las tripas de cómo está hecho sino simplemente se limitará a utilizarlo.

Siguiendo con el ejemplo del coche, cuando lo ponemos en marcha simplemente giramos la llave sin saber que hace dentro del capó. Lo mismo aplicado al código, tenemos una clase que dibuja un gráfico, éste tiene un método que realiza el dibujo, en nuestro programa llamaremos a éste método y ya está no sabemos como se las arregla para dibujar líneas, escribir, textos, rellenar superficies, ...

3.3 Polimorfismo

El polimorfismo se refiere a la posibilidad de definir múltiples clases con funcionalidad diferente, pero con métodos o propiedades denominados de forma idéntica, que pueden utilizarse de manera intercambiable mediante código cliente en tiempo de ejecución.

Otra vez un poco de paciencia, es mas fácil de lo que parece

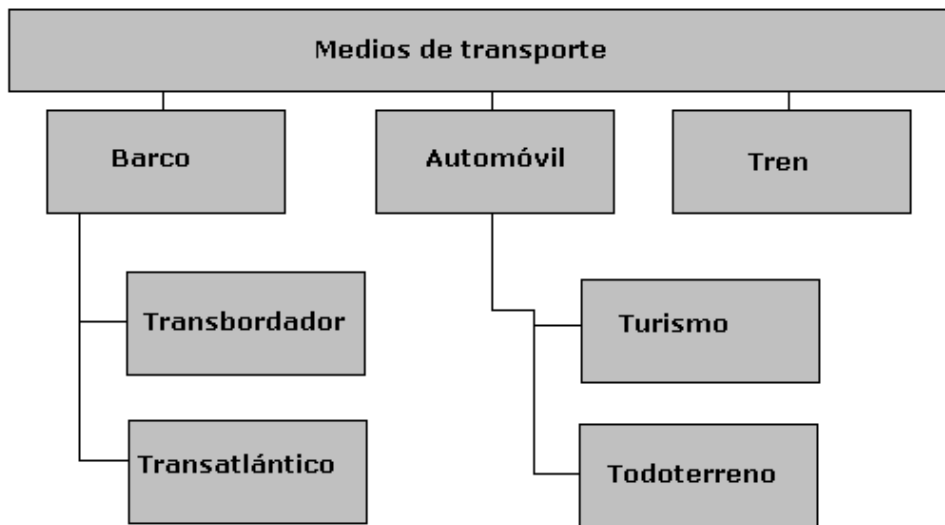
Programación orientada a objetos. Formularios Web

El Polimorfismo nos permite usar miembros de distintas clases de forma genérica sin tener que preocuparnos si pertenece a una clase o a otra. Siguiendo con el ejemplo de los animales, si el Gato y el Perro pueden morder podríamos tener un "animal" que muerda sin importarnos que sea el Gato o el Perro, simplemente podríamos usar el método Morder ya que ambos animales tienen esa característica "animal mordedor".

Dicho de otra forma, el polimorfismo determina que el mismo nombre de método realizará diferentes acciones según el objeto sobre el que se ha aplicado. Aplicado a la programación simplemente significa que en dos objetos: ventana y archivo al utilizar un mismo método se aplicará un resultado diferente. Es decir si aplicamos el método "Abrir" obviamente es el mismo nombre pero realiza cosas diferentes dependiendo del objeto que lo llame: en un caso se abrirá una ventana y en el otro un fichero.

4. Jerarquía de clases

Es una de las formas típicas de ver las superclases y subclases: en forma de árbol de clases o jerarquía:



Como ves el concepto es sencillo, la nomenclatura como hemos visto en otras ocasiones sería jerárquica: transportes.barco.transbordador para hacer referencia al nombre completo del objeto. En ocasiones si el nombre del objeto es único VB.NET es capaz de encontrar su sitio correcto, es decir, si en el código hacemos referencia a "turismo", como sólo existe un objeto con ese nombre en nuestra jerarquía de objetos no hay problema en trabajar con él sin escribir el nombre completo. En cambio si el nombre existe en varios sitios tendríamos que poner el nombre completo para romper esa ambigüedad.

4.1 Relaciones entre objetos

Los objetos se comunican entre sí mediante una serie de relaciones que vamos a comentar ahora:

4.2 Herencia

Ya vimos antes como era la característica más importante de la POO y permite crear una clase derivada de otra principal. Esta nueva clase tiene todos los métodos y propiedades que la original y además otros nuevos

que podemos añadir. En el código comprobaremos la relación de herencia "es un": "un todoterreno es un automóvil" es cierto pero "un todoterreno es un barco" devolvería falso.

4.3 Pertenencia

Los objetos pueden estar formados a su vez por otros objetos. Un objeto coche puede estar formado por un objeto "motor" o un objeto "dirección". En este caso hay una relación de pertenencia puesto que existe un conjunto de objetos que pertenecen a otro objeto o se unen para formar otro objeto.

En el código lo representaremos como "tiene un". Un "coche tiene un motor" sería cierto pero "una bicicleta tiene un motor" sería falso.

4.4 Utilización

En ocasiones un objeto utiliza a otro para alguna operación sin implicar ninguna pertenencia entre ellos. Por ejemplo un objeto Ventana puede utilizar un objeto Gráfico para mostrar el dibujo pero no es necesario que el objeto Grafico sea propiedad del objeto Ventana. La notación en el código es "usa un": el objeto Grafico usa una ventana sería cierto.

4.5 Reutilización

Finalmente si el objeto ha sido bien diseñado puede ser reutilizado por otra aplicación de forma directa o creando una clase derivada de él. Este es uno de los objetivos principales de la POO: aprovechar el código escrito.

Cambiemos de tema para ver cosas mas concretas, recordemos que las clases interactuarán con otras a través de tres ingredientes clave:

- **Propiedades:** Nos permiten acceder a los datos del objeto. Algunas serán de solo lectura y otras serán de escritura, por ejemplo en una variable de tipo string, una propiedad sería la longitud de la cadena que contiene
- **Métodos:** van a permitir realizar una acción en el objeto. Al contrario que las propiedades, los métodos realizan acciones, por ejemplo invocar el método de una matriz "sort" para que ordene sus elementos
- **Eventos:** Nos van a notificar de que sucede algo. Por ejemplo cuando se haga clic con el ratón sobre un botón.

5. Caso práctico

5.1 Crear o definir una clase

Al igual que existen instrucciones para declarar o definir una variable o cualquier otro elemento de un programa de Visual Basic, existen instrucciones que nos permiten crear o definir una clase.

Para crear una clase debemos usar la instrucción **Class** seguida del nombre que tendrá dicha clase, por ejemplo:

Programación orientada a objetos. Formularios Web

Class Cliente

A continuación escribiremos el código que necesitamos para implementar las propiedades y métodos de esa clase, pero para que Visual Basic sepa que ya hemos terminado de definir la clase, usaremos una instrucción de cierre:

End Class

Por tanto, todo lo que esté entre Class <nombre> y End Class será la definición de dicha clase.

Definir los miembros de una clase

Para definir los miembros de una clase, escribiremos dentro del "bloque" de definición de la clase, las declaraciones y procedimientos que creamos convenientes. Veamos un ejemplo:

```
Class Cliente

    Public Nombre As String

    Sub Mostrar()

        Response.Write("El nombre del cliente: {0}", Nombre)

    End Sub

End Class
```

En este caso, la línea **Public Nombre As String**, estaría definiendo una propiedad o "campo" público de la clase Cliente.

Por otro lado, el procedimiento Mostrar sería un método de dicha clase, en esta caso, nos permitiría mostrar la información contenida en la clase Cliente.

Esta es la forma más simple de definir una clase. Y normalmente lo haremos siempre así, por tanto podemos comprobar que es muy fácil definir una clase, así como los miembros de dicha clase.

Ahora vamos a crear objetos con esta clase:

Crear un objeto a partir de una clase

Como te he comentado antes, las clases definen las características y la forma de acceder a los datos que contendrá, pero sólo eso: los define. Para que podamos asignar información a una clase y poder usar los métodos de la misma, tenemos que crear un objeto basado en esa clase, o lo que es lo mismo: tenemos que crear una nueva instancia en la memoria de dicha clase. Para ello, haremos lo siguiente:

Definimos una variable capaz de contener un objeto del tipo de la clase, esto lo haremos como con cualquier variable:

```
Dim cli As Cliente
```

Pero, a diferencia de las variables basadas en los tipos visto hasta ahora, para poder crear un objeto basado en una clase, necesitamos algo más de código que nos permita "crear" ese objeto en la memoria, ya que con el código usado en la línea anterior, simplemente estaríamos definiendo una variable que es capaz de contener un objeto de ese tipo, pero aún no existe ningún objeto en la memoria,

Programación orientada a objetos. Formularios Web

para ello tendremos que usar el siguiente código:

```
cli = New Cliente()
```

Con esto le estamos diciendo al Visual Basic: crea un **nuevo** objeto en la memoria del tipo Cliente.

Estos dos pasos los podemos simplificar de la siguiente forma:

```
Dim cli As New Cliente()
```

A partir de este momento existirá en la memoria un objeto del tipo Cliente.

Nota: En las versiones anteriores de Visual Basic no era recomendable usar esta forma de instanciar un nuevo objeto en la memoria, porque, aunque de forma transparente para nosotros, el compilador añadía código extra cada vez que se utilizaba esa variable, pero en la versión .NET no existe ese problema y por tanto no deteriora el rendimiento de la aplicación.

Y ahora nos queda saber cómo acceder al objeto:

Acceder a los miembros de una clase

Para acceder a los miembros de una clase (propiedades o métodos) usaremos la variable que apunta al objeto creado a partir de esa clase, seguida de un punto y el miembro al que queremos acceder, por ejemplo, para asignar el nombre al objeto **cli**, usaremos este código:

```
cli.Nombre = "Guillermo"
```

Es decir, de la misma forma que haríamos con cualquier otra variable, pero indicando el objeto al que pertenece dicha variable. Como podrás comprobar, esto ya lo hemos estado usando anteriormente tanto en la clase Response como en las otras clases que hemos usado en temas anteriores, incluyendo los arrays (aunque todavía nos las entendamos muy bien).

Y para acceder al método Mostrar:

```
cli.Mostrar()
```

Ves que fácil.

Con lo visto este tema y este ejemplo queda bastante claro que son las propiedades y los métodos pero si tienes dudas los veremos en mas ocasiones... Sigamos, ahora veamos un poco la aplicación de la herencia.

5.2 Ejemplo de cómo usar la herencia

Para poder usar la herencia en nuestras clases, debemos indicar al Visual Basic que esa es nuestra intención, para ello disponemos de la instrucción **Inherits**, la cual se usa seguida del nombre de la clase de la que queremos heredar. Veamos un ejemplo.

Empezaremos definiendo una clase "base" la cual será la que heredaremos en otra clase.

Ya sabemos cómo definir una clase, aunque para este ejemplo, usaremos la clase Cliente que vimos anteriormente, después crearemos otra, llamada ClienteMoroso la cual heredará todas las características de la clase Cliente además de añadirle una propiedad a esa clase derivada de Cliente.

Veamos el código de estas dos clases. Para ello crea un nuevo proyecto del tipo consola y escribe estas líneas al principio o al final del fichero que el IDE añada de forma predeterminada.

```
Class Cliente
```

Programación orientada a objetos. Formularios Web

```
Public Nombre As String

Sub Mostrar()

    Response.Write("El nombre del cliente: {0}", Nombre)

End Sub

End Class

Class ClienteMoroso

    Inherits Cliente

    Public Deuda As Decimal

End Class
```

Como puedes comprobar, para que la clase ClienteMoroso herede la clase Cliente, he usado **Inherits Cliente**, con esta línea, (la cual debería ser la primera línea de código después de la declaración de la clase), le estamos indicando al VB que nuestra intención es poder tener todas las características que la clase Cliente tiene. Haciendo esto, añadiremos a la clase ClienteMoroso la propiedad Nombre y el método Mostrar, aunque también tendremos la nueva propiedad que hemos añadido: Deuda.

Ahora vamos a ver cómo podemos usar estas clases, para ello vamos a añadir código en el procedimiento Page_Load de la página:

```
Sub Page_Load()

    Dim cli As New Cliente()

    Dim cliM As New ClienteMoroso()

    '

    cli.Nombre = "Guille"

    cliM.Nombre = "Pepe"

    cliM.Deuda = 2000

    '

    Response.Write ("Usando Mostrar de la clase Cliente")

    cli.Mostrar()

    '

    Response.write ("Usando Mostrar de la clase ClienteMoroso")

    cliM.Mostrar()

    '

    Response.write ("La deuda del moroso es: {0}", cliM.Deuda)
```

Programación orientada a objetos. Formularios Web

```
End Sub
```

Lo que hemos hecho es crear un objeto basado en la clase Cliente y otro basado en ClienteMoroso. Le asignamos el nombre a ambos objetos y a la variable **cliM** (la del ClienteMoroso) le asignamos un valor a la propiedad Deuda.

Fíjate que en la clase ClienteMoroso no hemos definido ninguna propiedad llamada Nombre, pero esto es lo que nos permite hacer la herencia: heredar las propiedades y métodos de la clase base. Por tanto podemos usar esa propiedad como si la hubiésemos definido en esa clase. Lo mismo ocurre con los métodos, el método Mostrar no está definido en la clase ClienteMoroso, pero sí que lo está en la clase Cliente y como resulta que ClienteMoroso hereda todos los miembros de la clase Cliente, también hereda ese método.

La salida de este programa sería la siguiente:

```
Usando Mostrar de la clase Cliente
El nombre del cliente: Guille

Usando Mostrar de la clase ClienteMoroso
El nombre del cliente: Pepe
La deuda del moroso es: 2000
```

Ahora veamos cómo podríamos hacer uso del polimorfismo o al menos una de las formas que nos permite el .NET Framework.

Teniendo ese mismo código que define las dos clases, podríamos hacer lo siguiente:

```
Dim cli As Cliente

Dim cliM As New ClienteMoroso()
'

cliM.Nombre = "Pepe"

cliM.Deuda = 2000

cli = cliM
'

Response.write ("Usando Mostrar de la clase Cliente")

cli.Mostrar()
'

Response.write ("Usando Mostrar de la clase ClienteMoroso")

cliM.Mostrar()
'

Response.write ("La deuda del moroso es: {0}", cliM.Deuda)
```

Programación orientada a objetos. Formularios Web

En este caso, la variable `cli` simplemente se ha declarado como del tipo `Cliente`, pero no se ha creado un objeto, simplemente hemos asignado a esa variable el contenido de la variable `cliM`.

Con esto lo que hacemos es asignar a esa variable el contenido de la clase `ClienteMoroso`, pero como comprenderás, la clase `Cliente` "no entiende" nada de las nuevas propiedades implementadas en la clase derivada, por tanto sólo se podrá acceder a la parte que es común a esas dos clases: la parte heredada de la clase `Cliente`.

Realmente las dos variables apuntan a un mismo objeto, por eso al usar el método `Mostrar` se muestra lo mismo. Además de que si hacemos cualquier cambio a la propiedad `Nombre`, al existir sólo un objeto en la memoria, ese cambio afectará a ambas variables.

Para comprobarlo, añadimos este código al final...

```
cli.Nombre = "Juan"

Response.write ("Después de asignar un nuevo nombre a cli.Nombre")

cli.Mostrar()

cliM.Mostrar()
```

La salida de este nuevo código, sería la siguiente:

```
Usando Mostrar de la clase Cliente
El nombre del cliente: Pepe

Usando Mostrar de la clase ClienteMoroso
El nombre del cliente: Pepe
La deuda del moroso es: 2000

Después de asignar un nuevo nombre a cli.Nombre

El nombre del cliente: Juan

El nombre del cliente: Juan
```

La parte en gris es lo que se mostraría antes de realizar el cambio.

Como puedes comprobar, al cambiar en una de las variables el contenido de la propiedad `Nombre`, ese cambio afecta a las dos variables, pero eso no es porque haya nada mágico ni ningún fallo, es por lo que te comenté antes: sólo existe un objeto en la memoria y las dos variables acceden al mismo objeto, por tanto, cualquier cambio efectuado en ellas, se reflejará en ambas variables por la sencilla razón de que sólo hay un objeto en memoria.

A este tipo de variables se las llama variables por referencia, ya que hacen referencia o apuntan a un objeto que está en la memoria. A las variables que antes hemos estado viendo se las llama variables por valor, ya que cada una de esas variables tienen asociado un valor que es independiente de los demás. (esto ya lo vimos al ver las variables en los procedimientos)

Ejemplo

Vamos a crear una página de ejemplo creando una clase "calculadora". Este ejemplo va a ser una calculadora sencilla pero aplicando todo esto que estamos aprendiendo de los objetos.. Vamos a necesitar los siguientes

Programación orientada a objetos. Formularios Web

miembros:

- Propiedades: Valoractual
- Métodos: sumar, restar, multiplicar, dividir y borrar

También utilizaremos un miembro de datos privado que llamaremos "_actual" y que contendrá el valor expuesto por "valoractual"

1. Crea una página web con el nombre "clase1.aspx" y escribe el siguiente código en la sección que ves:

```
<script runat="server">

Public class calculadora

    private _actual as double

Public Readonly Property Valoractual as double

    Get

        return _actual

    End Get

End Property

Public Sub Suma (valor as double)

    _actual=_actual + valor

end Sub

Public Sub resta (valor as double)

    _actual=_actual - valor

end Sub

Public Sub Multiplica (valor as double)

    _actual=_actual * valor

end Sub

Public Sub Divide (valor as double)

    _actual=_actual / valor

end Sub
```


Programación orientada a objetos. Formularios Web

```
Public Sub Borra()  
    _actual=0  
end Sub  
  
End Class
```

Que es el código necesario para crear la clase. La primera línea es una directiva que indica que la página es de servidor "runat=server".

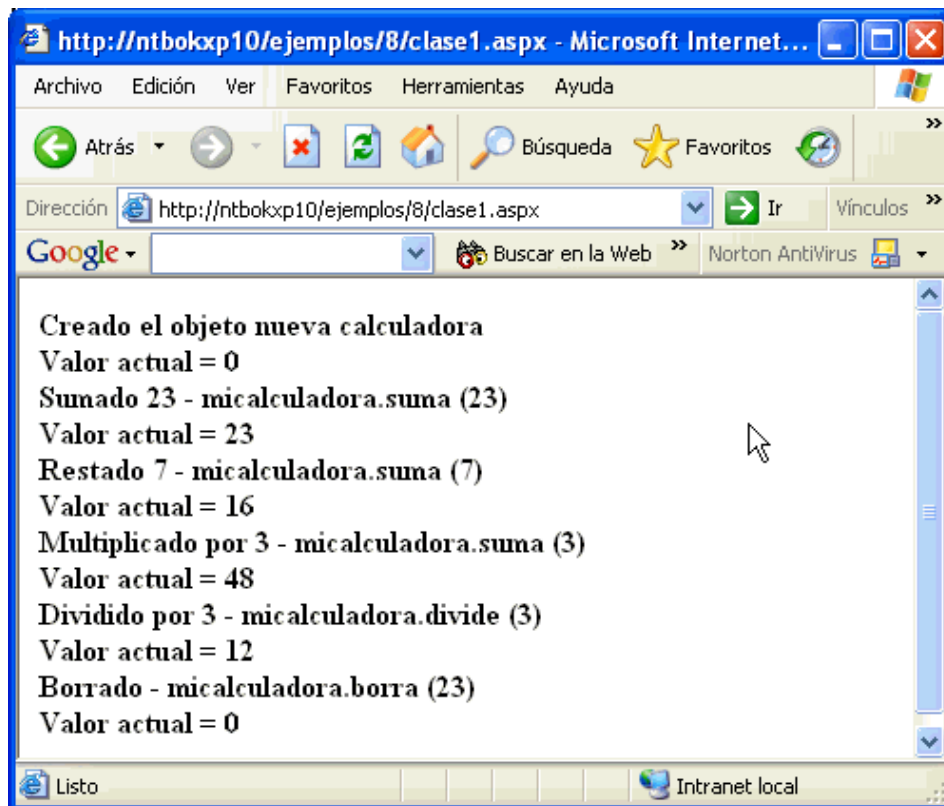
2. Y ahora creamos el procedimiento especial "Page_Load()" que se ejecutará al iniciarse la página. Por tanto hay dos secciones en esta página: una con la clase anterior y otra con este procedimiento para la escritura de resultados:

```
Sub Page_Load  
    Dim micalculadora as new calculadora ()  
  
    response.write ("<b>Creado el objeto nueva calculadora</b><br/>")  
    response.write ("<b>Valor actual = " & micalculadora.valoractual)  
  
    micalculadora.suma (23)  
    response.write ("<br/><b>Sumado 23 - micalculadora.suma (23)</b><br/>")  
    response.write ("<b>Valor actual = " & micalculadora.valoractual)  
  
    micalculadora.resta (7)  
    response.write ("<br/><b>Restado 7 - micalculadora.suma (7)</b><br/>")  
    response.write ("<b>Valor actual = " & micalculadora.valoractual)  
  
    micalculadora.multiplica (3)  
    response.write ("<br/><b>Multiplicado por 3 - micalculadora.suma (3)</b><br/>")  
    response.write ("<b>Valor actual = " & micalculadora.valoractual)  
  
    micalculadora.divide (4)  
    response.write ("<br/><b>Dividido por 3 - micalculadora.divide (3)</b><br/>")  
    response.write ("<b>Valor actual = " & micalculadora.valoractual)
```

Programación orientada a objetos. Formularios Web

```
micalculadora.borra  
  
response.write ("<br/><b>Borrado - micalculadora.borra (23)</b><br/>")  
  
response.write ("<b>Valor actual = " & micalculadora.valoractual)  
  
end Sub
```

3. Ejecutamos la página:



Que viene a resumir un poco todo lo visto hasta ahora en cuanto a las clases. Sigamos con mas detalles sobre las clases.

5.3 Sobrecargar el constructor de las clases

Una de las utilidades más prácticas de la sobrecarga de procedimientos es sobrecargar el constructor de una clase. Ahora veremos esto del constructor de la clase...

El constructor de una clase es un procedimiento de tipo Sub llamado New, dicho procedimiento se ejecuta cada vez que creamos un nuevo objeto basado en una clase. Si al declarar una clase no escribimos el "constructor", será el compilador de Visual Basic .NET el que se encargará de escribir uno genérico.

Esto es útil si queremos que al crear un objeto (o instancia) de una clase podamos hacerlo de varias formas, por ejemplo, sin indicar ningún parámetro o bien indicando algo que nuestra clase necesite a la hora de crear una nueva instancia de dicha clase.

Por ejemplo, si tenemos una clase llamada **Cliente**, puede sernos útil crear nuevos objetos indicando el nombre del cliente que contendrá dicha clase. Veamos con un ejemplo:

Programación orientada a objetos. Formularios Web

```
Class Cliente

    Public Nombre As String

    Public email As String

    '

    Sub New( )

        '

    End Sub

    Sub New(ByVal elNombre As String)

        Nombre = elNombre

    End Sub

End Class
```

Esta clase nos permite crear nuevos objetos del tipo **Cliente** de dos formas. Por ejemplo si tenemos una variable llamada **cli**, declarada de esta forma:

```
Dim cli As Cliente
```

podemos crear nuevas instancias sin indicar ningún parámetro:

```
cli = New Cliente()
```

o indicando un parámetro, el cual se asignará a la propiedad Nombre de la clase:

```
cli = New Cliente("Guillermo")
```

5.4 Los campos y las propiedades.

Recordamos que los campos son variables usadas a nivel de una clase: los campos representan los datos de la clase. Recuerda que la razón de ser de una clase es poder manipular cierta información: los campos y propiedades representan los datos manipulados por la clase y los métodos manipulan (o permiten manipular) esos datos.

Los campos representan la información que "internamente" la clase manipulará. Dependiendo de que esos campos sean accesibles sólo dentro de la clase o también puedan ser accedidos desde cualquier instancia, (creada en la memoria), de la clase, estarán declarados de una forma o de otra. Para simplificar, después entraremos en detalle, podemos declarar cualquier *miembro de una clase* de dos formas, según el nivel de visibilidad o ámbito que queramos que tenga.

Si lo declaramos con el modificador de acceso **Private**, ese miembro sólo será accesible desde "dentro" de la clase, es decir, en cualquier sitio de la clase podremos usar ese miembro, pero no será accesible desde "fuera" de la clase, por ejemplo, en una nueva instancia creada. Por otro lado, si declaramos un miembro de la clase como **Public**, ese miembro será accesible tanto desde dentro de la clase como desde fuera de la misma. Un miembro público de una clase siempre será accesible.

Nota: Cuando decimos <i>miembro de una clase</i> , me refiero a cualquier campo, propiedad, enumeración, método o evento.
--

Programación orientada a objetos. Formularios Web

Te aclaro que además del modificador Private y Public hay otros, que seguramente no veremos porque no nos van a ser útiles de momento en este curso...

Cuando declaramos un campo con el modificador Public, estamos haciendo que ese campo (o variable) sea accesible desde cualquier sitio, por otro lado, si lo declaramos como Private, sólo estará accesible en la propia clase.

Veamos un ejemplo. En el siguiente código vamos a declarar una clase que tendrá tres miembros públicos y uno privado. De estos tres miembros públicos, dos de ellos serán campos y el tercero será un método que nos permitirá mostrar por la consola el contenido de esos campos. El campo privado simplemente lo usaremos dentro del método, en otro ejemplo le daremos una utilidad más práctica, ya que en este ejemplo no sería necesario el uso de ese campo privado, pero al menos nos servirá para saber que "realmente" es privado y no accesible desde fuera de la clase.

```
Public Class Prueba14

    ' campo privado

    Private elNombreCompleto As String

    ' campos públicos

    Public Nombre As String

    Public Apellidos As String

    '

    ' método público

    Public Sub Mostrar()

        elNombreCompleto = Nombre & " " & Apellidos

        Response.Write(elNombreCompleto)

    End Sub

End Class

Page_Load

    ' creamos una nueva instancia de la clase

    Dim p As New Prueba14()

    '

    ' asignamos los valores a los campos públicos

    p.Nombre = "Guillermo"

    p.Apellidos = "Lopez"

    ' usamos el método para mostrar la información en la consola

    p.Mostrar()

    '

    ' esto dará error
```

Programación orientada a objetos. Formularios Web

```
'Response.write(p.elNombreCompleto)

End Sub

End Module
```

En la clase **Prueba14** (que está declarada como Public) tenemos declarado **Nombre** y **Apellidos** con el modificador Public, por tanto podemos acceder a estos dos campos desde una nueva instancia de la clase, así como desde cualquier sitio de la clase. Lo mismo es aplicable al método **Mostrar**, al ser público se puede usar desde la variable declarada en el procedimiento Page_Load.

Por otro lado, el campo **elNombreCompleto** está declarado como Private, por tanto sólo será accesible desde la propia clase, (esto se demuestra, porque se usa en el método Mostrar), y no desde fuera de ella, es decir, no podremos usar ese campo desde la instancia creada en Page_Load por la sencilla razón de que es "privada" y por tanto no visible ni accesible desde fuera de la propia clase. Si quitas el comentario que muestra por la consola el contenido del campo privado, comprobarás que dará un error en tiempo de compilación, ya que **elNombreCompleto** no es accesible por estar declarado como privado.

Si ya has trabajado antes con Visual Basic, a estos campos públicos también se les podían llamar propiedades, en Visual Basic .NET esto no ha cambiado y **se pueden considerar a los campos públicos como si fuesen propiedades de una clase**.

Pero, realmente, las propiedades son otra cosa diferente, al menos así deberíamos planteárnoslo y, tanto en Visual Basic .NET como en las versiones anteriores, además de declarar una propiedad usando la declaración de un campo público, también podemos usar la instrucción **Property** para que quede más claro cual es nuestra intención.

Como veremos a continuación, el uso de Property nos dará algo de más trabajo, pero también nos proporcionará mayor control sobre lo que se asigne a esa propiedad. Por ejemplo, imagínate que no se debería permitir que se asigne una cadena vacía al Nombre o al Apellido, en el ejemplo anterior, esto no sería posible de "controlar", ya que al ser "variables" públicas no tenemos ningún control sobre cómo ni cuando se asigna el valor, por tanto, salvo que creemos unos métodos específicos para asignar y recuperar los valores de esas dos "propiedades", no tendríamos control sobre la asignación de una cadena vacía.

Vamos a ver cómo se podría "controlar" esto que acabo de comentar, para simplificar, sólo vamos a comprobar si al nombre se le asigna una cadena vacía, para ello, vamos a crear un procedimiento que se encargue de asignar el valor al nombre y otro que se encargue de recuperar el contenido de esa "propiedad".

Estos métodos se llamarán **AsignaNombre** y **RecuperaNombre**. El primero nos permitirá asignar un nuevo nombre y el segundo nos permitirá recuperar el contenido de esa "propiedad".

```
Public Class Prueba14_2

    ' variable (campo) privado para guardar el nombre

    Private elNombre As String

    '

    ' campo público para los apellidos

    Public Apellidos As String

    '

    ' método que permite asignar el nuevo nombre

    Public Sub AsignarNombre(ByVal nuevoNombre As String)

        ' comprobamos si no es una cadena vacía

        If nuevoNombre <> "" Then

            elNombre = nuevoNombre

        End If

    End Sub

End Class
```

Programación orientada a objetos. Formularios Web

```
End If

End Sub

' método que recupera el nombre asignado

Public Function RecuperarNombre() As String

    Return elNombre

End Function

'

' método público para mostrar el nombre completo

Public Sub Mostrar()

    Response.Write(elNombre & " " & Apellidos)

End Sub

End Class

Page_Load

    ' creamos una nueva instancia de la clase

    Dim p As New Prueba14_2()

    '

    ' asignamos los valores a los campos públicos

    p.AsignarNombre("Guillermo")

    p.Apellidos = "Lopez"

    ' usamos el método para mostrar la información en la consola

    p.Mostrar()

    '

    Response.write("El nombre es: " & p.RecuperarNombre)

    '

End Sub
```

Aunque pueda parecer una "barbaridad" (casi lo es), analicemos el contenido de la clase Prueba14_2 para ver si nos enteramos de las cosas...

Declaramos una variable privada (un campo privado) llamado **elNombre**, este campo nos permitirá almacenar el nombre que se asigne usando el método **AsignarNombre**, el cual recibirá un parámetro, antes de asignar el contenido indicado en ese parámetro, comprobamos si es una cadena vacía, de ser así, no hacemos nada, ya que sólo se asigna el nuevo valor si el contenido del parámetro es distinto de una cadena vacía, con esto conseguimos lo que nos proponíamos: no asignar al nombre una cadena vacía.

Programación orientada a objetos. Formularios Web

Por otro lado, si queremos acceder al contenido del "nombre", tendremos que usar el método **RecuperarNombre**, esta función simplemente devolverá el contenido del campo privado **elNombre**.

En cuanto al campo **Apellidos**, (el cual actuará como una propiedad, es decir, representa a un dato de la clase), éste permanece sin cambios, por tanto no se hace ninguna comprobación y podemos asignarle lo que queramos, incluso una cadena vacía.

Esto está muy bien, pero no es "intuitivo", lo lógico sería poder usar la propiedad **Nombre** tal como la usamos en el primer ejemplo, pero de forma que no permita asignarle una cadena vacía. Esto lo podemos conseguir si declaramos **Nombre** como un procedimiento de tipo **Property**. La forma de usarlo sería como en el segundo ejemplo, pero en lugar de usar **AsignarNombre** o **RecuperarNombre**, usaremos **Nombre**, que es más intuitivo, aunque, como comprobarás, "internamente" es como si usáramos los dos procedimientos que acabamos de ver.

5.5 ¿Cómo declarar una propiedad como un procedimiento Property?

Para estos casos, el compilador de Visual Basic .NET pone a nuestra disposición una instrucción, que al igual que Sub o Function, nos permiten declarar un procedimiento que tiene un trato especial, este es el caso de **Property**.

La forma de usar Property es muy parecido a como se declara una función, pero con un tratamiento especial, ya que dentro de esa declaración hay que especificar por un lado lo que se debe hacer cuando se quiera recuperar el valor de la propiedad y por otro lo que hay que hacer cuando se quiere asignar un nuevo valor.

Cuando queremos recuperar el valor de una propiedad, por ejemplo para usarlo en la parte derecha de una asignación o para usarlo en una expresión, tal es el caso de que queramos hacer algo como esto:

```
Dim s As String = p.Nombre
```

O como esto otro:

```
Response.write (p.Nombre)
```

En estos dos casos, lo que queremos es recuperar el contenido de la propiedad. Pero si lo que queremos es asignar un nuevo valor, esa propiedad normalmente estará a la izquierda de una asignación, como sería el caso de hacer esto: **p.Nombre = "Guillermo"**. En este caso estaríamos asignando un nuevo valor a la propiedad Nombre.

Vale, esto está muy bien, pero... ¿cómo se declara un procedimiento Property? Vamos a esto...

Si queremos que **Nombre** sea realmente una propiedad (un procedimiento del tipo Property) para que podamos hacer ciertas comprobaciones tanto al asignar un nuevo valor como al recuperar el que ya tiene asignado, tendremos que crear un procedimiento como el siguiente:

```
Public Property Nombre() As String

    ' la parte Get es la que devuelve el valor de la propiedad
    Get

        Return elNombre

    End Get

    ' la parte Set es la que se usa al asignar el nuevo valor
    Set(ByVal Value As String)

        If Value <> "" Then

            elNombre = Value
```

Programación orientada a objetos. Formularios Web

```
End If  
  
End Set  
  
End Property
```

Es decir, declaramos un procedimiento del tipo **Property**, que tiene dos bloques internos:

- El primero es el bloque **Get**, que será el código que se utilice cuando queramos recuperar el valor de la propiedad, por ejemplo para usarlo en la parte derecha de una asignación o en una expresión.
- El segundo es el bloque **Set**, que será el código que se utilice cuando queramos asignar un nuevo valor a la propiedad, tal sería el caso de que esa propiedad estuviera en la parte izquierda de una asignación.

Como puedes comprobar, el bloque Set recibe un parámetro llamado **Value** que es del mismo tipo que la propiedad, en este caso de tipo String. Value representa el valor que queremos asignar a la propiedad y representará lo que esté a la derecha del signo igual de la asignación. Por ejemplo, si tenemos esto: **p.Nombre = "Guillermo"**, "Guillermo" será lo que Value contenga.

Fíjate que al declarar la propiedad, no se indica ningún parámetro, esto lo veremos en otra ocasión, pero lo que ahora nos interesa saber es que lo que se asigna a la propiedad está indicado por el parámetro Value del bloque Set.

Nota: En el caso de Visual Basic .NET, el parámetro indicado en el bloque Set se puede llamar como queramos. En C#, siempre se llamará **value**, además de que no se indica el parámetro en el bloque **set**. Por esa razón, se recomienda dejar el nombre Value, que es el que el VB .NET utiliza automáticamente cuando declaramos un procedimiento de tipo Property.

Fíjate que cuando creamos un procedimiento Property siempre será necesario tener un campo (o variable) privado que sea el que contenga el valor de la propiedad. Ese campo privado lo usaremos para devolver en el bloque Get el valor de nuestra propiedad y es el que usaremos en el bloque Set para conservar el nuevo valor asignado.

lógicamente el tipo de datos del campo privado debe ser del mismo tipo que el de la propiedad.

La ventaja de usar propiedades declaradas como Property en lugar de usar variables (o campos) públicos es que podemos hacer comprobaciones u otras cosas dentro de cada bloque Get o Set, tal como hemos hecho en el ejemplo de la propiedad Nombre para que no se asigne una cadena vacía al Nombre.

De todas formas, no es recomendable hacer mucho "trabajo" dentro de una propiedad, sólo lo justo y necesario. Si nuestra intención es que dentro de una propiedad se ejecute un código que pueda consumir mucho tiempo o recursos, deberíamos plantearnos crear un método, ya que las propiedades deberían asignar o devolver los valores de forma rápida.

Debido a que en Visual Basic .NET los campos públicos son tratados como propiedades, no habría demasiada diferencia en crear una propiedad declarando una variable pública o usando un procedimiento Property, pero deberíamos acostumbrarnos a crear procedimientos del tipo Property si nuestra intención es crear una propiedad, además de que el uso de procedimientos Property nos da más juego que simplemente declarando una variable pública.

Nota: En las versiones de Visual Basic anteriores a .NET la forma de declarar una propiedad es mediante tres tipos de procedimientos Property:

- Property Get para devolver el valor de la propiedad,
- Property Let para asignar el nuevo valor a la propiedad y
- Property Set para asignar un objeto a esa propiedad.

En Visual Basic .NET sólo existen dos bloques: Get y Set, el segundo sería el equivalente a los dos últimos de Visual Basic 6.0

5.6 Propiedades de sólo lectura.

Una de las ventajas de usar un procedimiento Property es que podemos crear propiedades de sólo lectura, es decir, propiedades a las que no se pueden asignar valores nuevos, simplemente podemos acceder al valor que contiene.

Programación orientada a objetos. Formularios Web

Para poder conseguir que una propiedad sea de sólo lectura, tendremos que indicárselo al Visual Basic .NET de la siguiente forma:

```
Private valorFijo As Integer = 10
'

Public ReadOnly Property Valor() As Integer

    Get

        Return valorFijo

    End Get

End Property
```

Es decir, usamos la palabra clave (o modificador) `ReadOnly` al declarar la propiedad y tan sólo especificamos el bloque `Get`. Si declaramos un procedimiento `ReadOnly Property` no podemos indicar el bloque `Set`, eso dará error.

5.7 Propiedades de sólo escritura.

De la misma forma que podemos definir una propiedad de sólo lectura, también podemos crear una propiedad de sólo escritura, es decir, una propiedad que sólo aceptará que se asignen nuevos valores, pero que no permitan obtener el valor que tienen... la verdad es que este tipo de propiedades no son muy habituales, pero podemos hacerlo. Veamos cómo tendríamos que declarar una propiedad de sólo escritura.

```
Private valorEscritura As Boolean
'

Public WriteOnly Property Escribir() As Boolean

    Set(ByVal Value As Boolean)

        valorEscritura = Value

    End Set

End Property
```

Es decir, usamos el modificador `WriteOnly` al declarar la propiedad y sólo debemos especificar el bloque `Set`. Si declaramos un procedimiento `WriteOnly Property` no podemos indicar el bloque `Get`, ya que eso dará error.

Cuando declaramos una propiedad de sólo lectura no podemos declarar otra propiedad con el mismo nombre que sólo sea de escritura. Si nuestra intención es crear una propiedad de lectura/escritura, simplemente con no complicarnos la existencia es suficiente, es decir, declaramos la propiedad sin indicar ni `ReadOnly` ni `WriteOnly`.

En las versiones anteriores de Visual Basic si sólo queríamos una propiedad de sólo lectura, simplemente se declaraba el procedimiento **Property Get** y no era necesario indicar `ReadOnly`. El mero hecho de no declarar el procedimiento **Property Let** indica que es de sólo lectura. Esto mismo también es aplicable a C#, en ese lenguaje no es necesario indicar explícitamente si es o no de sólo lectura. Por tanto, si ya has programado con C# y quieres hacer algo en VB .NET tendrás que tener esto presente.

5.8 Campos de sólo lectura.

Lo mismo que existen propiedades de sólo lectura, podemos crear campos de los que sólo podamos leer el valor que contiene y no asignar ninguno nuevo. Aunque en principio podrías pensar que con una constante se podría conseguir lo mismo que con un campo de

Programación orientada a objetos. Formularios Web

solo lectura, como verás puede haber una pequeña diferencia de la que nos podemos aprovechar.

Para aclarar esto último, te diré, aunque ya lo sabes, que una constante es como una variable que tiene un valor fijo, es decir, una vez que se ha asignado, no se puede cambiar. Los campos de sólo lectura a diferencia de las constantes, se pueden cambiar de valor, pero sólo en la definición, lo cual no se diferenciaría de la forma de declarar una constante, o dentro del constructor de la clase.

Esto último es algo que no se puede hacer con una constante, ya que **las constantes siempre tienen el mismo valor**, el cual se asigna al declararla.

Veamos a ver un par de ejemplos. En el siguiente código vamos a declarar una constante y también un campo (o variable) de sólo lectura.

```
Public Const PI As Double = 3.14159

Public ReadOnly LongitudNombre As Integer = 50
```

En este código tenemos declarada una constante llamada **PI** que tiene un valor fijo. Las constantes siempre deben declararse con el valor que contendrán.

Por otro lado, tenemos un campo de sólo lectura llamado **LongitudNombre**, que es del tipo Integer y tiene un valor de 50. Como podrás comprobar, en este caso no hay diferencia entre una constante y un campo de sólo lectura. La verdad es que si lo dejamos así, no habría ninguna diferencia entre una declaración y otra.

Bueno, si hay diferencia. Cuando declaramos una constante pública, ésta estará accesible en las nuevas instancias de la clase además de ser accesible "globalmente", es decir, no tendremos que crear una nueva instancia de la clase para poder acceder a la constante. Por tanto podríamos decir que las constantes declaradas en una clase son "variables" compartidas por todas las instancias de la clase. Es como si declarásemos la constante usando **Shared** o como si estuviese declarada en una clase de tipo Module, como ya te comenté anteriormente, la diferencia entre una clase de tipo Module y una de tipo Class es que en la primera, todos los miembros están compartidos (Shared), mientras que en la segunda, salvo que se indique explícitamente, cada miembro pertenecerá a la instancia de la clase, es decir, de cada objeto creado con New. De esto hablaremos más en profundidad en otra ocasión.

Suponte que cambiamos la declaración de LongitudNombre de la siguiente forma:

```
Public Shared ReadOnly LongitudNombre As Integer = 50
```

En este caso, no habría diferencia con una constante.

Pero, esta no sería la forma habitual de declarar un campo de sólo lectura. Lo habitual es declararlo sin un valor inicial, aunque haciéndolo así nos aseguramos que tenga un valor predeterminado, en caso de que no se asigne ninguno nuevo.

Como te comenté hace unas líneas, la forma de asignar el valor que tendrá un campo de sólo lectura, sería asignándolo en el constructor de la clase. Por tanto, podríamos tener un constructor (Sub New) que reciba como parámetro el valor que tendrá ese campo de sólo lectura.

En el siguiente código vamos a declarar una clase que tendrá un campo de sólo lectura, el cual se asigna al crear una nueva instancia de la clase.

```
Public Class Prueba14_6

    Public ReadOnly LongitudNombre As Integer = 50

    '

    Public Sub New()

        '

    End Sub

End Class
```

Programación orientada a objetos. Formularios Web

```
Public Sub New(ByVal nuevaLongitud As Integer)

    LongitudNombre = nuevaLongitud

End Sub

'
' Este será el punto de entrada del ejecutable

Public Shared Sub Main()

    '
    ' si creamos la clase sin indicar la nueva longitud...

    Dim p As New Prueba14_6()

    ' el valor será el predeterminado: 50

    Response.write("p.LongitudNombre = {0}", p.LongitudNombre)

    '
    ' si creamos la clase sin indicar la nueva longitud...

    Dim p1 As New Prueba14_6(25)

    ' el valor será el indicado al crear la instancia

    Response.write("p1.LongitudNombre = {0}", p1.LongitudNombre)

End Sub

End Class
```

Esta clase tiene definidos dos constructores: uno sin parámetros y otro que recibe un valor de tipo Integer, ese valor será el que se use para el campo de sólo lectura. En el Sub Main, el cual está declarado como Shared para que se pueda usar como punto de entrada del ejecutable, declaramos dos objetos del tipo de la clase, el primero se instancia usando New sin ningún parámetro, mientras que el segundo se crea la nueva instancia indicando un valor en el constructor, ese valor será el que se utilice para darle valor al campo de sólo lectura, cosa que se demuestra en la salida del programa:

```
p.LongitudNombre = 50

p1.LongitudNombre = 25
```

En los comentarios está aclarado porqué el objeto **p** toma el valor 50 y porqué usando **p1** el valor es 25.

Y para que queden las cosas claras, decirte que una vez que hemos asignado el valor al campo de sólo lectura, ya no podemos modificar dicho valor, salvo que esa modificación se haga en el constructor. Por tanto, sólo podemos asignar un nuevo valor a un campo de sólo lectura en el constructor de la clase. Dicho esto, si nuestras mentes fuesen tan retorcidas como para hacer esto, sería factible y no produciría ningún error:

```
Public Sub New(ByVal nuevaLongitud As Integer)

    LongitudNombre = nuevaLongitud

    LongitudNombre = LongitudNombre + 10

End Sub
```

Programación orientada a objetos. Formularios Web

Por la sencilla razón de que "dentro" del constructor podemos asignar un nuevo valor al campo de sólo lectura.

Si has conseguido llegar hasta aquí ENHORABUENA, acabas de recibir y terminar un intensivo de programación orientada a objetos. Esto como os dije al principio es obligatorio que lo viésemos, ya lo siento pero era necesario. Ahora vamos a aplicar todo esto en nuestras páginas ASP así que vamos a ver la última parte de teoría que son los objetos en ASP.NET

5.9 Pasos por valor y referencia

Tipos de datos por valor

Cuando dimensionamos una variable, por ejemplo, de tipo Integer, el CLR reserva el espacio de memoria que necesita para almacenar un valor del tipo indicado; cuando a esa variable le asignamos un valor, el CLR almacena dicho valor en la posición de memoria que reservó para esa variable. Si posteriormente asignamos un nuevo valor a dicha variable, ese valor se almacena también en la memoria, sustituyendo el anterior, eso lo tenemos claro, ya que así es como funcionan las variables. Si a continuación creamos una segunda variable y le asignamos el valor que contenía la primera, tendremos dos posiciones de memoria con dos valores, que por pura casualidad, resulta que son iguales, entre otras cosas porque a la segunda hemos asignado un valor "igual" al que tenía la primera.

Vale, seguramente pensarás que eso es así y así es como te lo he explicado o al menos, así es como lo habías entendido. Pero, espera a leer lo siguiente y después seguimos hablando... a ver si te parece que lo tenías "aprendido".

Tipos de datos por referencia

En Visual Basic .NET también podemos crear objetos, los objetos se crean a partir de una clase, (del tema de las clases hablaremos un poco mas adelante). Cuando dimensionamos una variable cuyo tipo es una clase, simplemente estamos creando una variable que es capaz de manipular un objeto de ese tipo, en el momento de la declaración, el CLR no reserva espacio para el objeto que contendrá esa variable, ya que esto sólo lo hace cuando usamos la instrucción New.

En el momento en que creamos el objeto, (mediante New), es cuando el CLR reserva la memoria para dicho objeto y le dice a la variable en que parte de la memoria está almacenado, de forma que la variable pueda acceder al objeto que se ha creado. Si posteriormente declaramos otra variable del mismo tipo que la primera, tendremos dos variables que saben "manejar" datos de ese tipo, pero si a la segunda variable le asignamos el contenido de la primera, en la memoria no existirán dos copias de ese objeto, sólo existirá un objeto que estará referenciado por dos variables. Por tanto, cualquier cambio que se haga en dicho objeto se reflejará en ambas variables.

Vamos a ver un par de ejemplos para aclarar esto de los tipos por valor y los tipos por referencia.

Crea un nuevo proyecto de tipo consola y añade el siguiente código:

```
Sub Page_Load()  
  
    ' creamos una variable de tipo Integer  
  
    Dim i As Integer  
  
    ' le asignamos un valor  
  
    i = 15  
  
    ' mostramos el valor de i  
  
    Response.write ("i vale " & i)  
  
    ,
```

Programación orientada a objetos. Formularios Web

```
' creamos otra variable

Dim j As Integer

' le asignamos el valor de i

j = i

Response.write ("<br>")

Response.write ("hacemos esta asignación: j = i")

'

' mostramos cuanto contienen las variables

Response.write ("i vale " & i & " y j vale " & j)

'

' cambiamos el valor de i

i = 25

Response.write ("<br>")

Response.write("hacemos esta asignación: i = 25")

' mostramos nuevamente los valores

Response.write ("<br>")

Response.write("i vale " & i & " y j vale " & j)

'

End Sub
```

Como puedes comprobar, cada variable tiene un valor independiente del otro. Esto está claro.

Ahora vamos a ver qué es lo que pasa con los tipos por referencia. Para el siguiente ejemplo, vamos a crear una clase con una sola propiedad, ya que las clases a diferencia de los tipos por valor, deben tener propiedades a las que asignarles algún valor.

No te preocupes si no te enteras todavía mucho de todo esto, ahora sólo es para ver con un ejemplo esto de los tipos por referencia.

```
Class prueba

Public Nombre As String

End Class

Sub Form_Load()

' creamos una variable de tipo prueba

Dim a As prueba

' creamos (instanciamos) el objeto en memoria
```

Programación orientada a objetos. Formularios Web

```
a = New prueba()  
  
' le asignamos un valor  
  
a.Nombre = "hola"  
  
' mostramos el contenido de a  
  
Response.write("a vale " & a.Nombre)  
  
'  
  
' dimensionamos otra variable  
  
Dim b As prueba  
  
'  
  
' asignamos a la nueva el valor de a  
  
b = a  
  
Response.write ("<br>")  
  
Response.write ("hacemos esta asignación: b = a")  
  
'  
  
' mostramos el contenido de las dos  
  
Response.write ("<br>")  
  
Response.write("a vale " & a.nombre & " y b vale " & b.Nombre)  
  
' cambiamos el valor de la anterior  
  
a.Nombre = "adios"  
  
  
Response.write ("<br>")  
  
Response.write("hacemos una nueva asignación a a.Nombre")  
  
'  
  
' mostramos nuevamente los valores  
  
Response.write ("<br>")  
  
Response.write("a vale " & a.nombre & " y b vale " & b.Nombre)  
  
'  
  
End Sub
```

La clase **prueba** es una clase muy simple, pero como para tratar de los tipos por referencia necesitamos una clase, he preferido usar una creada por nosotros que cualquiera de las clases que el .NET Framework nos ofrece.

Dimensionamos una variable de ese tipo y después creamos un nuevo objeto del tipo **prueba**, el cual asignamos a la variable **a**.

Programación orientada a objetos. Formularios Web

Una vez que tenemos "instanciado" (o creado) el objeto al que hace referencia la variable **a**, le asignamos a la propiedad **Nombre** de dicho objeto un valor. Lo siguiente que hacemos es declarar otra variable del tipo **prueba** y le asignamos lo que contiene la primera variable.

Hasta aquí, es casi lo mismo que hicimos anteriormente con las variables de tipo Integer. La única diferencia es la forma de manipular las clases, ya que no podemos usarlas "directamente", porque tenemos que crearlas (mediante New) y asignar el valor a una de las propiedades que dicha clase contenga. Esta es la primera diferencia entre los tipos por valor y los tipos por referencia, pero no es lo que queríamos comprobar, así que sigamos con la explicación del código mostrado.

Cuando mostramos el contenido de la propiedad **Nombre** de ambas variables, las dos muestran lo mismo, que es lo esperado, pero cuando asignamos un nuevo valor a la variable **a**, al volver a mostrar los valores de las dos variables, ¡las dos siguen mostrando lo mismo!

Y esto no es lo que ocurría en el primer ejemplo.

¿Por qué ocurre? Porque las dos variables, apuntan al mismo objeto que está creado en la memoria.

¡Las dos variables hacen referencia al mismo objeto! y cuando se realiza un cambio mediante una variable, ese cambio afecta también a la otra, por la sencilla razón que se está modificando el único objeto de ese tipo que hay creado en la memoria.

Para simplificar, los tipos por valor son los tipos de datos que para usarlos no tenemos que usar New, mientras que los tipos por referencia son tipos (clases) que para crear un nuevo objeto hay que usar la instrucción New.

6. Espacios de nombres (namespace) y ensamblados (assemblies)

Al principio de este mismo tema comentamos algo de los "namespaces" o espacios de nombres, veámoslo ahora con un poco más de detalle. El núcleo de .NET ya sabemos que consiste en una gran biblioteca de clases y objetos que manipularemos de la forma que hemos aprendido en la primera parte de este capítulo. .NET incluye cientos de clases que nos van a ofrecer una potencia y versatilidad inusitada, parece mentira que una página web pueda admitir tanta programación. Los antiguos programadores de ASP y de Visual Basic hemos encontrado un mundo absolutamente nuevo en .NET, todas las carencias y limitaciones de estos lenguajes han desaparecido y además .NET después de ya varios años en el mercado se ha consolidado como una plataforma madura y estable. (difícil decir esto de Microsoft)

Estas clases nos van a permitir:

- Trabajar con páginas web
- Acceso a bases de datos
- Generación de gráficos
- Monitorizar rendimientos del servidor
- ...

Prácticamente lo que pienses hacer en programación existe en .NET.

6.1 Espacios de nombres (namespace)

Ya hemos hablado en otras ocasiones de los espacios de nombres o "namespaces". .NET utiliza un esquema de nombres para **organizar todas las clases que contiene**, de esta forma al estar agrupadas por temas es más fácil localizarlas, piensa en un "namespace" como un grupo de clases. Por ejemplo existe un grupo de clases para acceder a bases de datos que se encuentran en el espacio de nombres "System.Data". Las clases

Programación orientada a objetos. Formularios Web

necesarias para la creación de páginas web se encuentran en "System.Web".

Los espacios de nombres son una estructura jerárquica donde "System" es la raíz y define los principales bloques. Por ejemplo las definiciones mas importantes como "boolean" y "string" están dentro de este bloque. System.Web contiene clases genéricas para la creación de sitios web mientras que el siguiente nivel System.Web.UI contiene las clases para crear las interfaces de las páginas web como los botones.

Cuando hacemos referencia a una clase de un espacio de nombres utilizamos la sintaxis

```
<espacio de nombres>.<nombre de clase>
```

Por ejemplo la clase que se utiliza para trabajar con ficheros se llama "File" y está dentro del espacio de nombre "System.IO" así que para hacer referencia a esta clase pondríamos:

```
System.IO.File
```

También podemos extender esta sintaxis para hacer referencia a las propiedades o métodos de las clases, por ejemplo para abrir un archivo podríamos:

```
System.IO.File.Open ( "nombrefichero.txt" )
```

Para simplificar esta sintaxis podemos incluir espacios de nombres en nuestros programas de esta forma "importamos" ese espacio de nombres y no tenemos que escribir toda la ruta completa. Así que utilizaremos la instrucción "imports" en nuestro programa para añadir ese grupo de clases a nuestro programa, de esta forma sólo tendríamos que escribir:

```
File.Open ( "nombrefichero.txt" )
```

Hay varias razones para agrupar las clases en espacios de nombres:

- Organizar. Hay montones de clases, si las organizamos en subconjuntos jerárquicos la estructura lógica será muy sencilla de tratar
- Universalidad. Podemos crear nuestras propias clases y añadirlas a un espacio de nombres con la posibilidad de hacer un "import" de esas clases en nuestros programas
- Facilidad de uso. Una vez importadas las clases con el "Import" la sintaxis es mucho mas sencilla.

Espacios de nombres en ASP.NET

Todas las clases relacionadas con las páginas web están agrupadas en el espacio de nombres "System.Web". Las clases mas importantes están dentro del espacio de nombres que hemos dicho pero hay otras mas especializadas que están dentro de otras secciones, veamos:

- System.Web. Proporciona todas las clases básicas para crear páginas web. Este espacio de nombres se añade automáticamente a nuestras páginas ASPX, de ahí que nunca hayamos tenido que incluirlo.
- System.Web.UI. Proporciona las clases para crear formularios HTML, por ejemplo los botones y cuadros de texto.
- System.IO. Incluye clases para trabajar con ficheros. Ojo no con bases de datos, sino con ficheros.
- System.Collections. Las colecciones son objetos que contienen listas de objetos, con este "namespace" podremos trabajar con colecciones
- System.Diagnostics. Proporciona las clases para diagnosticar problemas.

- System.Data. Para trabajar con bases de datos
- System.Drawing. Clases para crear gráficos de forma dinámica.
- System.XML. Para trabajar con datos XML.

6.2 Utilizar namespace

Normalmente nuestro proyecto ya incorporará los espacios de nombres necesarios para empezar a trabajar. Pero si queremos organizar nuestro código en un spacename podemos definir uno utilizando una estructura como esta:

```
Namespace Miempresa  
  
Namespace Miprograma  
  
Public Class Producto  
    'Código...  
  
End Class  
  
End Namespace  
  
End Namespace
```

En este ejemplo la clase Producto es del espacio de nombres Miempresa.Miprograma. El código dentro de estos espacios de nombres se accede directamente por el nombre, desde fuera habría que poner el nombre cualificado completo, es decir: Miempresa.Miprograma.Producto

Ni que decir tiene que esto nos servirá para incorporar nuestros programas o funciones a otro proyecto mayor garantizando la reutilización del código. Como ves la función de los espacios de nombres es contener código para ayudarnos a organizar nuestras clases.

6.3 Importar namespace

Habitualmente importaremos los predefinidos por .NET a nuestros programas pero si queremos añadir uno como el creado anteriormente simplemente pondríamos:

```
Imports Miempresa.Miprograma
```

Si no hubiésemos importado este espacio de nombres una declaración de un objeto de este espacio de nombres se convertiría en:

```
Dim ventas_producto as New Miempresa.Miaplicacion.Producto()
```

Obviamente será mucho mas manejable si lo hemos importado, fíjate la diferencia:

```
Dim ventas_producto as New Producto()
```

La importación de espacios de nombres es un tema organizativo, no influye en el rendimiento de la aplicación.

6.4 Ensamblados (Assemblies)

Ya sabemos la utilidad que tiene la posibilidad de utilizar las bibliotecas de clases, namespaces, en un programa .NET. Físicamente.. ¿cómo se representan estas bibliotecas? Buena pregunta, sabemos crear espacios de nombres y sabemos importar lo que ya trae .NET pero... ¿dónde se encuentran? La respuesta está en los ensamblados o assemblies, son ficheros físicos que contienen el código compilado. Normalmente tienen una extensión .exe como aplicaciones individuales o .DLL si son componentes reutilizables.

La relación no es estricta ya que un ensamblado puede contener varios espacios de nombres. Digamos que así como los espacios de nombres es la forma lógica de agrupar clases, los ensamblados son el paquete físico para distribuir nuestro código o programas. Luego parece que el objetivo de Visual Basic .NET o de ASP.NET es producir un ensamblado con nuestra aplicación, pues si, eso parece, pero sigamos viendo mas cosas...

Las clases .NET se encuentran por tanto en varios archivos físicos, por ejemplo la "System" se encuentra en el fichero "mscorlib.dll". Muchas de las utilizadas por ASP.NET se encuentran en el ensamblado "System.Web".

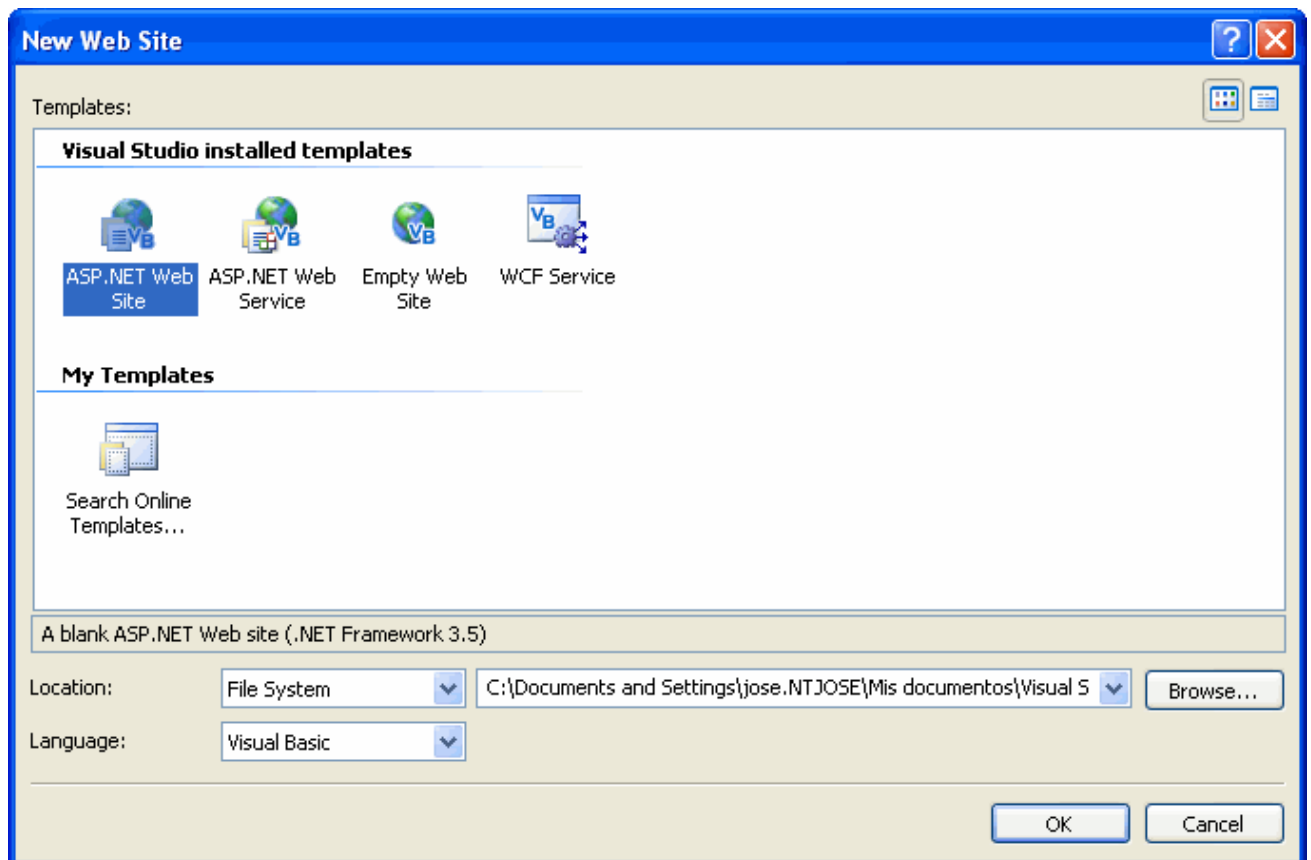
Cuando compilemos nuestro programa tendremos que decirle a nuestro compilador los ensamblados que necesita, pero por defecto se incorporan prácticamente los que necesitamos, así que será una labor sencilla. Pero si necesitamos un ensamblado de otro fabricante, por ejemplo para poder crear PDF's en nuestras páginas web, tendremos que incorporar ese ensamblado mediante una referencia a él, pero esto ya lo veremos mas adelante.

Y ahora por fin empezamos con el entorno de desarrollo para trabajar ya todo en él.

7. Desarrollar aplicaciones en ASP.NET

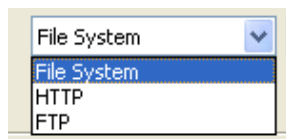
7.1 El entorno de desarrollo integrado.

Comencemos por conocer el entorno de desarrollo integrado, IDE. Ya lo hemos utilizado en unas sencillas aplicaciones aisladas ahora vamos a conocerlo mas a fondo. Comencemos creando otro sitio web limpio:



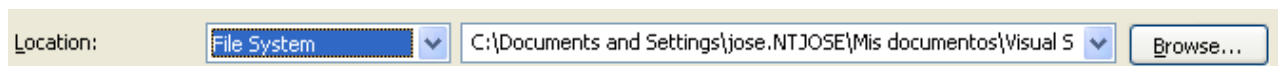
Ya la conocemos de antes, así que seleccionamos crear un sitio web, en la ayuda que pone justo debajo podemos ver que será de .NET Framework 3.5, es decir de la versión 2.008. Si estuviéramos trabajando con Visual Studio completo, en lugar del Web developer que está centrado sólo en proyectos WEB tendríamos algunas opciones mas, como por ejemplo decirle con que versión de .NET queremos trabajar. Aunque en nuestro caso lo tenemos claro, con la última y mas potente, la 3.5

Especificaremos una ubicación para nuestro proyecto dependiendo de si elegimos:



Fíjate la diferencia de almacenarlo en distintos sitios:

- File System. Es decir una carpeta de nuestro disco, por tanto el botón "browse" explorará nuestras unidades



- HTTP. Los crea en un servidor web. Ojo, tiene que tener instaladas las extensiones de FrontPage, que ya vimos en el capítulo 2. Fíjate que en esta opción, igual que en la siguiente, ya no será nuestro equipo el que ejecute las páginas sino un servidor que tenga lógicamente el IIS instalado.

Location:	HTTP	http://servidor	Browse...
-----------	------	-----------------	-----------

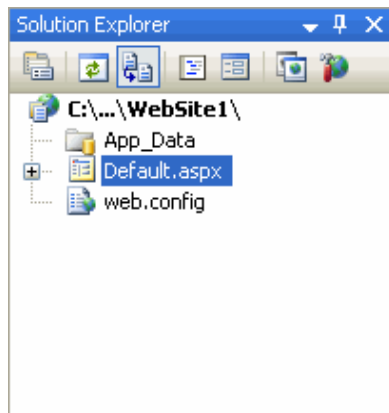
- FTP. Crea el sitio web en un servidor de ficheros FTP. Este equipo deberá tener instalado IIS para poder ejecutar las páginas. Es muy habitual que se trabaje en local y luego mediante FTP accedamos a nuestro servidor externo de internet para actualizar las nuevas páginas.

Location:	FTP	ftp://servidor	Browse...
-----------	-----	----------------	-----------

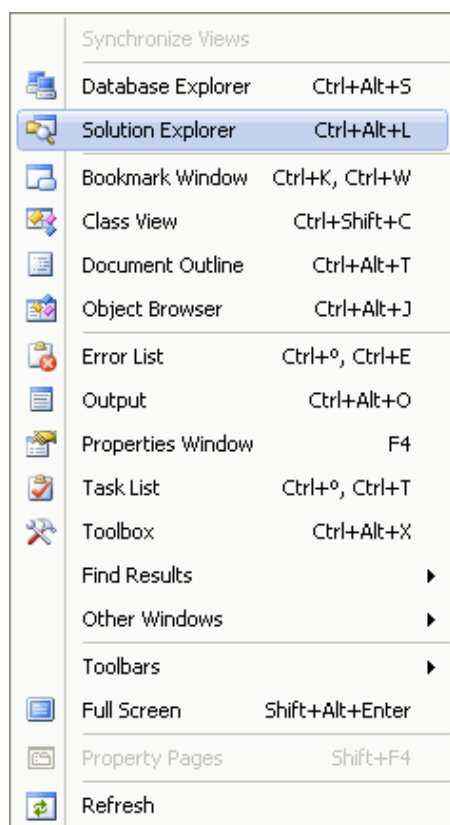
Finalmente pulsaremos "OK" con la opción que hayamos seleccionado. Si estamos creando un sitio web local simplemente nos lo creará en nuestro directorio y sino se encargará de mantenerlo en el servidor web o FTP. Al finalizar nos aparecerá una página predeterminada con código HTML, como ya vimos por encima capítulos atrás.

7.2 El explorador de soluciones

Gráficamente está en la parte superior derecha:



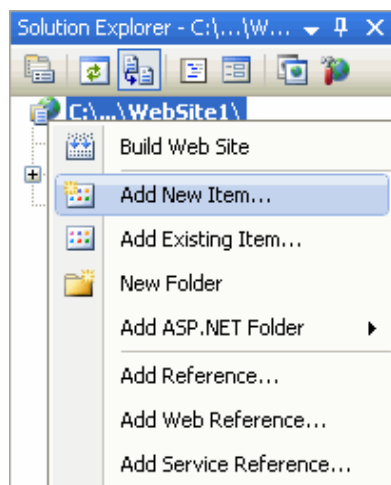
Si no te aparece en pantalla, selecciona en la opción de menú ver "View" la opción "Solution Explorer".



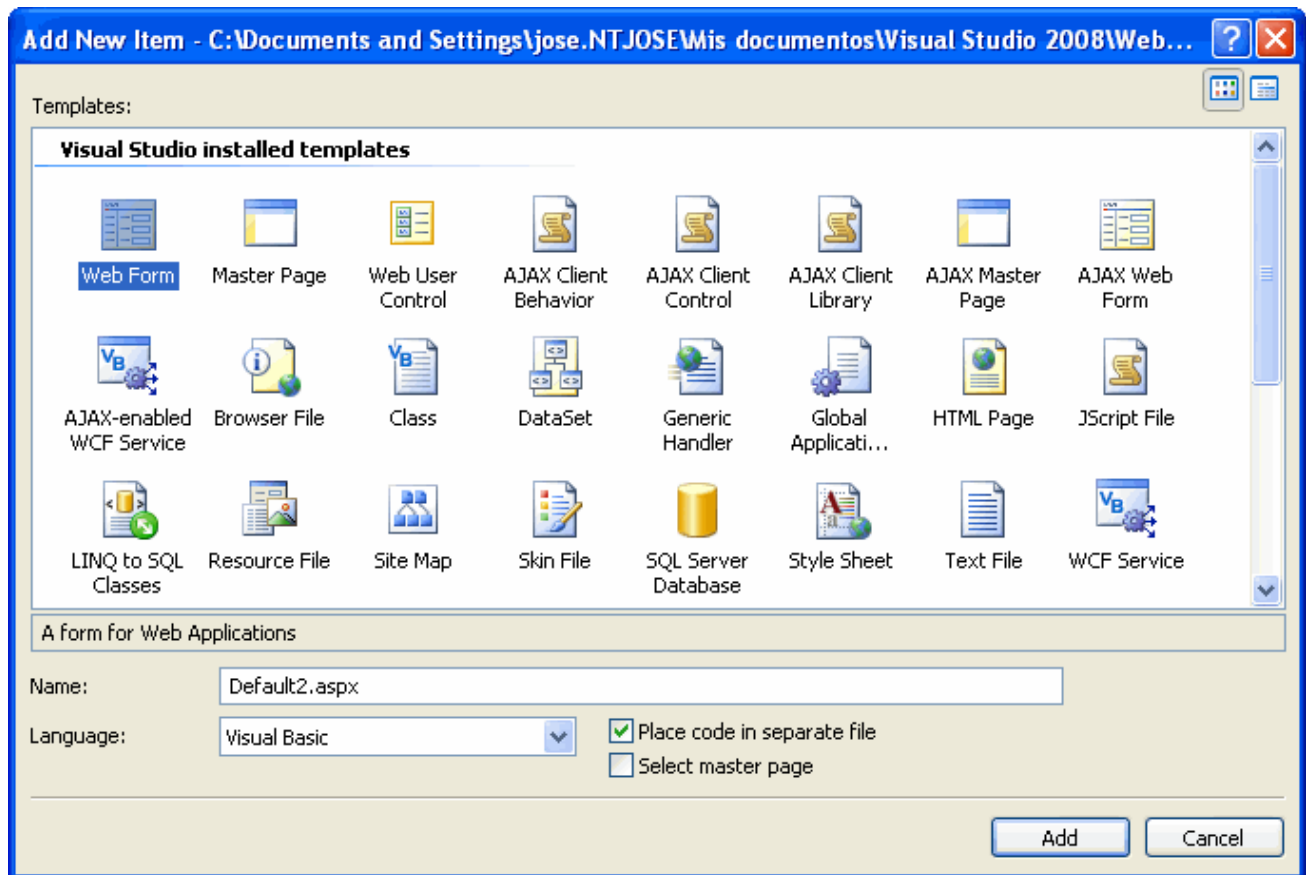
Esta vista nos mostrará todos los ficheros de que consta nuestro sitio web. Mejor dicho, muestra todos los ficheros de la carpeta donde se almacena nuestro sitio web. Por ejemplo, si copiamos un fichero a esa carpeta desde el explorador de archivos de Windows, nos aparecería en esta lista al actualizar la vista.

7.3 Añadir formularios Web

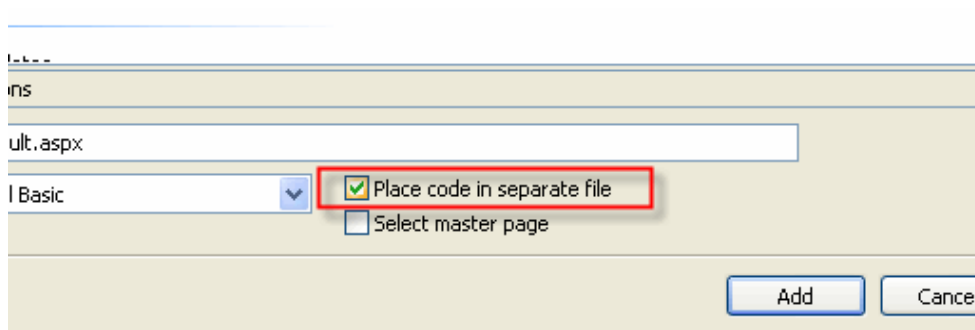
A lo largo de nuestro proyecto tendremos que añadir mas páginas web, para esto podemos seleccionar "New Item" desde el menú "WebSite" o con el botón derecho en el título de nuestro web en el explorador de soluciones:



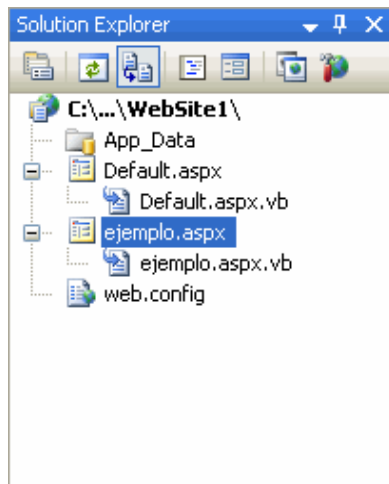
Tendremos a nuestra disposición varios tipos distintos de formularios:



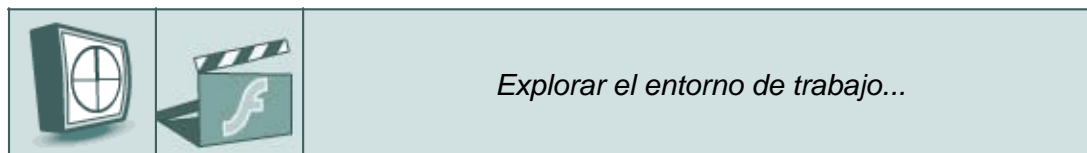
No esta mal la variedad ¿no?. De momento utilizaremos los "Web Form" que serán nuestras páginas ASP.NET estándar. Hemos visto en los ejemplos anteriores a este capítulo que en las páginas mezclábamos código HTML con ASP.NET. Esto lo hacíamos porque dejamos desactivada esta opción:



Aunque nos ha venido muy bien para ilustrar nuestros sencillos ejemplos no es la técnica que vamos a utilizar. Utilizaremos la técnica de "código atrás", es decir, tendremos una página web con el código HTML y otra paralela con el código ASP.NET, así no mezclamos el código y es mas fácil de mantener, aunque inicialmente te pueda desconcertar algo. El rendimiento es el mismo, es mas para organizar el código, que en páginas complejas sería muy complicado de mantener si lo tuviéramos todo mezclado. Veamos un ejemplo, añade un ejemplo de un formulario web que le llamaremos "ejemplo.aspx":

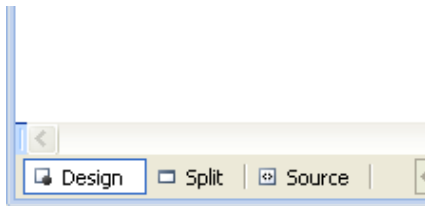


Como ves nos ha creado la página llamada "ejemplo.aspx" y ampliando el árbol vemos que ha creado otra con el mismo nombre pero con la extensión .vb: ejemplo.aspx.vb. Para terminar podemos añadir páginas o elementos ya existentes con la opción "Add Existing Item" desde las mismas opciones de menú, en el de arriba del "website" y en el contextual del explorador de soluciones.



7.4 Diseñar una página web

Ahora que ya conocemos un poco de cómo se organiza nuestro sitio web vamos a diseñar una sencilla página. Seleccionaremos nuestra página de ejemplo "default.aspx" que como ya sabes es la página donde comienza nuestro sitio web. Ábrela y fíjate en las tres vistas posibles:



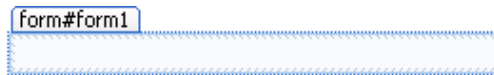
- En la vista "Diseño" tendremos la representación gráfica de la página que estamos diseñando
- En "Source" veremos el código HTML y ASP.NET de la página
- La vista "Split" nos divide la ventana en dos partes para ver el código y su representación gráfica de forma simultánea.

No utilizaremos una visión en particular ya que alternaremos las tres vistas según en la fase del diseño de la página estemos. Una vez será todo visual y otras escribiremos largas líneas de código en la vista "Source".

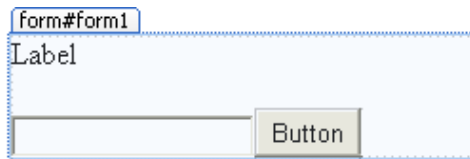
Añadir componentes web

Para añadir un control web ASP.NET simplemente lo seleccionaremos de la lista de la izquierda y lo arrastraremos hasta nuestra página. Te habrás fijado en la gran cantidad de controles que ASP.NET pone a

nuestra disposición, a lo largo de este curso veremos la mayoría de ellas. De todas formas el arrastre de los controles puede que no lo hagamos correctamente, tendremos que tener mucho cuidado de que estos controles se encuentran dentro de las etiquetas que delimitan un formulario "<form>". Si no está dentro de esta sección el control no funcionará. Así que asegúrate de ponerlos dentro de la zona del formulario:



Los controles están agrupados por tipos: estándar, de validación, de acceso a datos, para navegación. Vamos a practicar añadiendo un control "label", un cuadro de texto y un botón:



Abre ahora la vista "split" para ver el código que el IDE nos va colocando:

```
1  <%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
2
3  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://ww
4
5  <html xmlns="http://www.w3.org/1999/xhtml">
6  <head runat="server">
7      <title>Untitled Page</title>
8  </head>
9  <body>
10     <form id="form1" runat="server">
11         <asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
12         <br />
13         <br />
14         <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
15         <asp:Button ID="Button1" runat="server" Text="Button" />
16     </form>
17 </body>
18 </html>
```

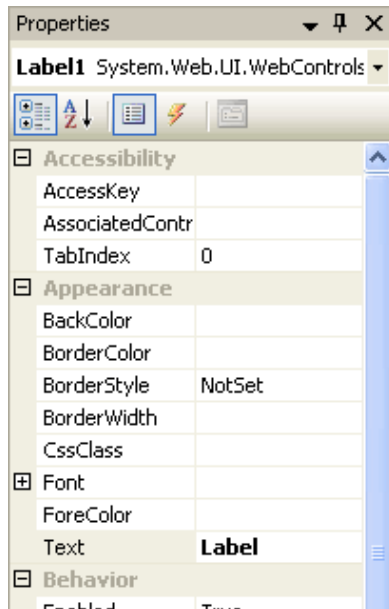
Hay una serie de etiquetas estándar como <html> y <body>. A continuación tenemos las etiquetas que delimitan el formulario <form> y luego los tres controles que hemos añadido:

- Etiqueta: <asp:label>
- Cuadro de texto: <asp:TextBox>
- Botón: <asp:Button>

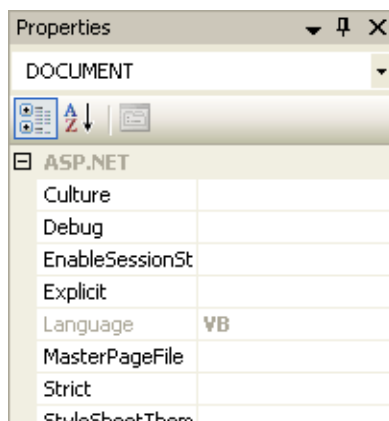
Así que ten en cuenta esto: los controles que añadamos deben estar siempre entre las etiquetas de un formulario. Si nos dejamos alguno fuera no mandará los datos de ese control.

La ventana Propiedades

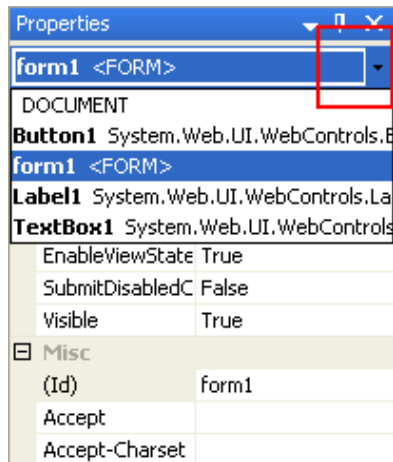
Hemos añadido unos controles pero ahora debemos modificar su contenido para escribir los textos que queramos. Algún dato, o propiedad, de estos controles los podremos escribir haciendo clic sobre el propio control pero necesitaremos ver todas sus propiedades para tener un control completo sobre él. Para eso haremos clic en el control y observaremos en la parte derecha inferior la ventana "propiedades":



Como ves podemos cambiar el texto del control con la propiedad "text", el color del texto y otras propiedades importantes. Si esta ventana no está visible la puedes activar desde el menú "view" y luego "Properties Window". Tenla siempre visible, la utilizaremos mucho en nuestro trabajo con los formularios. Practica un poco con los controles de la etiqueta (Label) y el cuadro de texto (TextBox) que acabamos de añadir al formulario. Ten en cuenta que si hacemos clic en medio de la página fuera de los controles se activarán las propiedades de la página:



Como ves el objeto que aparece seleccionado es "DOCUMENT" que es la propia página. Si haces clic dentro del área del formulario, donde hemos colocado nuestros controles verás que se seleccionan las propiedades de éste en la ventana propiedades. Finalmente si haces clic en la flecha a la derecha del elemento seleccionado en propiedades:



Verás una lista de todos los elementos que componen la página web. Familiarízate con esta ventana de propiedades y a la vez que cambies un elemento del control fíjate en las propiedades que se van añadiendo en la vista del código. Por ejemplo, voy a poner textos en los dos controles, cambiar el tamaño de uno de ellos y un color, el resultado de:

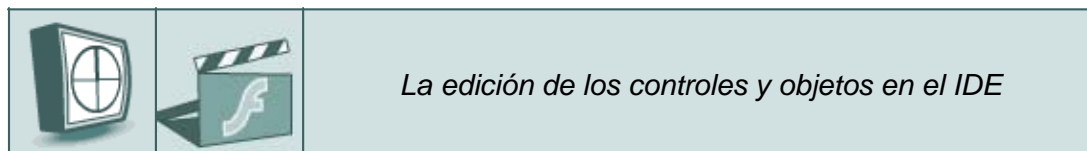
Etiqueta de prueba

Texto de prueba	Enviar datos
-----------------	--------------

En HTML será:

```
<form id="form1" runat="server">
<asp:Label ID="Label1" runat="server" Font-Size="Large" Height="21px"
    Text="Etiqueta de prueba" Width="238px"></asp:Label>
<br />
<asp:TextBox ID="TextBox1" runat="server" Width="251px">Texto de prueba</asp:TextBox>
<asp:Button ID="Button1" runat="server" Text="Enviar datos" Width="124px" />
</form>
```

Como ves, a la definición de cada objeto (<asp:label) le sigue una lista de propiedades, como el tipo de letra, tamaño, texto, estos son las propiedades del objeto que iremos modificando de la ventana.



8. Formularios Web

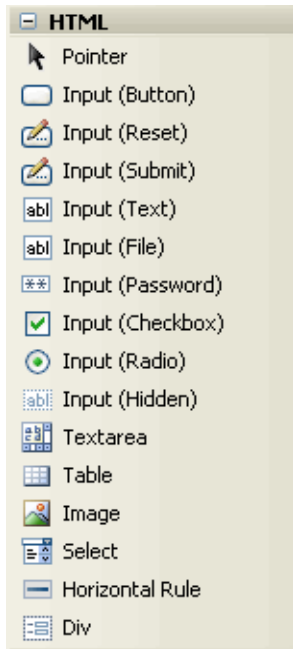
Ya hemos hablado bastante de los formularios, vamos por fin a definirlos y a trabajar con ellos. Los formularios o "Web Form" agrupan dos bloques distintos de código:

- La plantilla HTML que contiene los controles de servidor ASP.NET. Es el responsable de la presentación del formulario web en el navegador

Programación orientada a objetos. Formularios Web

- El código ASP.NET necesario para proporcionar la lógica de proceso. Es decir para realizar las consultas a las bases de datos y otros procesos, genera dinámicamente el contenido que se mostrará. El contenido de este resultado normalmente se presenta con otros controles de servidor.

Muy importante, hay dos grupos de controles que podremos poner en nuestras páginas los control HTML estándar, que en la ventana de controles de la izquierda los tienes bajo el grupo "HTML":



Y los controles ASP.NET que son los restantes que ves en esa misma ventana. Los controles ASP.NET son los mismos que los estándar HTML pero enormemente mejorados y sobre todo, ampliados, ya que tenemos decenas de ellos. Imagina que trabajas con un lenguaje de programación que tiene 10 controles para construir las páginas, acceder a bases de datos, mostrar los datos... y tienes otro con 40 controles mucho mas versátiles y potentes, pues esta es la diferencia entre ellos.

Como la ejecución siempre va a ser en el servidor, siempre me va a devolver código HTML que el navegador no tendrá ningún problema en mostrar. Ya irás encajando todas las piezas a medida que vayamos haciendo ejemplos.

Te resumo entonces los tres tipos de páginas que podemos encontrar:

- **Páginas Web.** Estáticas, contiene información diversa
- **Formularios HTML.** Utilizando HTML estándar y los controles de formulario crea páginas para enviar información
- **Web Forms.** Son formularios que utilizan unos controles de servidor mas potentes que los anteriores y que muestran un resultado. Todo hecho con ASP.NET

Cuando comencemos a utilizar ASP.NET utilizaremos los controles de servidor ASP.NET. Éstos incluyen todos los controles HTML pero con mas funciones y bastante mas avanzados. Recuerda que como todo se ejecuta en el servidor lo que va a recibir el usuario en su navegador será código estándar, la potencia de ASP.NET está en el proceso en el servidor.

Programación orientada a objetos. Formularios Web

Un control de servidor es aparentemente igual que el control de formulario HTML pero produce un "objeto" de servidor es decir un componente de programación avanzado que nos ofrecerá muchas mas posibilidades.

Sigamos con la definición del formulario, sabemos que el lenguaje HTML se compone de etiquetas que engloban secciones, por ejemplo las etiquetas

```
<table>

</table>
```

Indican el comienzo y final de una tabla: </> es el carácter que indica que termina el componente html. Otras etiquetas escriben títulos, por ejemplo:

```
<h1> Título de la lección </h1>
```

Escribe ese texto en letras grandes. Bien, para diferenciar los controles estándar de los de ASP.NET se pone la sintaxis:

```
<asp:Button>
```

Que define un control de ASP para el botón. De todas formas ya no hay necesidad, como antes se hacía, de escribir todo esto a mano, porque como ya hemos comprobado, nuestro IDE nos escribirá este código por nosotros. Volvamos a fijarnos en el código generado en el ejemplo anterior:

```
1  <%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb" Inherits="_Default" %>
2
3  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
4
5  <html xmlns="http://www.w3.org/1999/xhtml">
6  <head runat="server">
7      <title>Untitled Page</title>
8  </head>
9  <body>
10     <form id="form1" runat="server">
11         <asp:Label ID="Label1" runat="server" Font-Size="Large" Height="21px"
12             Text="Etiqueta de prueba" Width="238px"></asp:Label>
13         <br />
14         <asp:TextBox ID="TextBox1" runat="server" Width="251px">Texto de prueba</asp:TextBox>
15         <asp:Button ID="Button1" runat="server" Text="Enviar datos" Width="124px" />
16     </form>
17 </body>
18 </html>
19 |
```

Verás las etiquetas que delimitan el formulario <form> y </form> y dentro de él los controles que hemos añadido. Vamos a aclarar algún detalle, por ejemplo el efecto de este código en un navegador.

Estamos diciendo que esto se ejecuta en el servidor: cierto, ASP.NET siempre se ejecuta en el servidor y que al cliente (navegador) se le envía código HTML estándar. Por tanto se traducirán esos controles asp a controles estándar HTML cuando se envíen los datos al navegador. Aun así lo verás mas claro enseguida, cuando hagamos ejemplos. Veamos ahora que significa la primera fila de la página de ejemplo...

8.1 La Directiva de página

```
<% @ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb" Inherits="_Default" %>
```

En todos los formularios web de ASP.NET nos encontraremos con un encabezado parecido a este que contiene información sobre la página. Puedes ver que define que el lenguaje va a ser VB y no C#, que tiene una página con el código asociada llamada "default.aspx.vb" y algún parámetro más. Es importante conocer estos detalles de las páginas porque nos da mucha información de cómo va a ser su comportamiento.

Por supuesto, al tratarse de cosas de nuestro IDE, no irán al navegador del usuario.

8.2 El tipo de documento

El tipo de documento es otra sección de la página que verás a continuación de la anterior:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

En páginas que no son de ASP.NET esta etiqueta figura en la primera línea y en nuestras páginas y como puedes ver, la segunda posición. Esta etiqueta indica el tipo de lenguaje de hipertexto que se está utilizando, por ejemplo HTML ó XML. Esta etiqueta es opcional pero se añade porque así proporcionan información a los navegadores para mejorar la interpretación de las páginas.

En nuestro caso le está indicando que utilizaremos XHTML 1.0 y además la web donde está definido este lenguaje estándar. Por fin Microsoft no ha hecho "su estándar". ¿Así que no estamos utilizando HTML, sino XHTML? Pues sí, XHTML es una ampliación estándar del HTML y por tanto con mejoras. Sobre todo en la validación de las etiquetas escritas que como primera mejora va a producir un código más consistente que hará que las páginas se vean iguales en todos los navegadores. Antes el HTML no era tan flexible y una misma página se veía distinta en un navegador (Internet Explorer) y otro (Firefox).

Nuestro ASP funcionará igual, la diferencia estará en el código HTML generado ya que será más sólido utilizando la variante XHTML.

No vamos a detenernos en explicarte los distintos componentes XHTML, ya que los iremos creando en nuestras páginas, sólo te enumeraré algunos:

- <table> </table>. Crea una tabla
- H1, H2, H3, H4, H5 y H6 crean titulares de mayor a menor tamaño: <h1> título de la página </h1>
- <p> Indica el comienzo de un párrafo
-
 introduce una línea en blanco
- para incluir una imagen en la página web.
- Crea un hipervínculo
- ...

Por supuesto cada etiqueta tendrá sus propios atributos o propiedades, por ejemplo para una imagen:

```

```

Nos colocará en la página el gráfico "icono.gif" y le dará la propiedad "alt" que hará que cuando pasemos el ratón por encima del gráfico nos saldrá una ventana de información que pondrá "Hola". La sintaxis de los controles permite hacer la simplificación que ves gracias a que al final se pone el carácter / que indica que termina la etiqueta. Es decir, esto:

```
</img>
```

es lo mismo que el ejemplo:

```

```

Ya que esa barra del final "/" le indica que ahí termina el control. Es una sintaxis que verás habitualmente en este curso.

8.3 Página web completa

Después de la sección del formato de la página con "DOCTYPE" que acabamos de ver viene una sección llamada `<head>` que contiene elementos de la cabecera de la página web y que habitualmente será el propio título de la página. Finalmente el `<body>` nos engloba el contenido de la página web, por tanto una página completa sería:

```
<html xmlns="http://www.w3.org/1999/xhtml">

<head runat="server">

    <title>Untitled Page</title>

</head>

<body>

</body>

</html>
```

9. Escribir código

Ya conocemos las partes de una página web con su lenguaje HTML, que por suerte nos lo escribirá prácticamente todo nuestro Visual Web Developer, pero por lo menos ya sabremos qué es cada parte y podremos tocar sin miedo algún elemento de la página si se diese el caso.

Para escribir código ya sabemos que utilizaremos la vista código o "code" con la que ya hemos practicado. Pero ahora tendremos que ir por fin a la página asociada para escribir el código. Si recuerdas al crear una página se crea una con la extensión `aspx` donde pondremos nuestros controles y el código HTML y otra donde pondremos nuestro código ASP. Así separamos los dos mundos y será mas fácil de mantener. Fíjate en el explorador de soluciones como tenemos la página asociada a nuestra `default.aspx`:



Vamos a hacer clic en esa segunda página para conocerla:

```
1 |  
2 Partial Class _Default  
3     Inherits System.Web.UI.Page  
4  
5 End Class  
6
```

De momento tiene este código de ejemplo y además si te fijas en la pantalla no hay mas vistas, es decir, esta no tienen una vista de código y otra de diseño "desing" ya que sólo va a contener código ASP.NET, por tanto no hay que colocar controles dentro de ella, todos esos van en la página estándar. Para movernos por estas páginas que vamos abriendo fíjate que las tenemos en las solapas de arriba:



Así que podremos cambiarnos rápidamente de una a otra. Sigamos con nuestra página de código ASP.NET que como ves es bastante simple. Simplemente importa el espacio de nombres estándar de ASP.NET, recuerda que al ver los espacios de nombres comentamos que este es el estándar para los formularios web.

De momento nos vale con este pero está claro que añadiremos mas ya que por ejemplo cuando empecemos a trabajar con SQL Server como servidor base de datos tendremos que importar el espacio de nombres correspondiente para proporcionar a nuestra página y entorno todos los controles y objetos para el acceso a datos.

En esta página incluiremos todo nuestro código como por ejemplo los métodos de los objetos (una consulta a una bbdd) y manejaremos los eventos de los objetos, por ejemplo que hacer cuando se pulse en un botón.

9.1 Controladores de eventos

El concepto de la programación orientada a eventos es muy fácil de asimilar. Trasladémonos a un programa Windows, tenemos un programa con un formulario, cuando hacemos clic en un cuadro desplegable automáticamente se rellena otro con unos datos, al pulsar en una casilla de verificación se cambiar de color un texto y al pulsar en el botón "Aceptar" se produce una consulta a una base de datos. Todas estas cosas que hemos hecho han disparado unos eventos que se enlazan con una programación.

Es decir al pulsar en un botón se activa el evento "click" del botón con lo que podremos colocar código en ese evento para que realice una acción. Al desplazarnos por una lista de un cuadro desplegable se dispara o se activa el evento "onchange" que nos permite escribir el código necesario para que se ejecute una serie de instrucciones.

En las páginas web es difícil encontrar esta sincronización porque lo que tenemos en pantalla es un código HTML que se nos muestra en el navegador, no es un programa que tiene el control de los eventos.

Sin embargo una de la características de ASP.NET y sólo de este lenguaje es un acercamiento bastante grande a los eventos de un auténtico programa Windows. Podemos llamar a procedimientos que se ejecuten al dispararse o activarse un evento. Además el colocar el código en los eventos que se producen en el programa ayuda a modularizar el código y organizar la escritura de procedimientos y funciones. Comencemos ya con la programación orientada a eventos.

¿Qué es un evento?

Un ejemplo muy sencillo, estamos trabajando en nuestra mesa y suena el teléfono con un determinado mensaje. Eso es un evento, el programa está realizando su trabajo o esperando alguna acción y cuando se activa un evento (llamada de teléfono) la atendemos para realizar otras tareas (atenderla). Dependiendo de la solicitud que hagan en ese evento (la llamada puede pedir unos datos o la realización de un pedido) ejecutaremos una serie de instrucciones.

Podemos decir que se pueden producir de distintas formas:

- Se produce un evento (llamada de teléfono)
- El sistema detecta un evento (escucha la llamada)
- El sistema realiza una acción (contestar a la llamada)

En Windows sucede lo mismo: un usuario hace clic con el ratón. El ratón manda un mensaje al sistema operativo. Éste recibe el mensaje del ratón y genera el evento "clic". El programa activo detecta el evento y realiza las operaciones necesarias: muestra un menú, cierra un formulario...

¿Qué es la programación orientada a eventos?

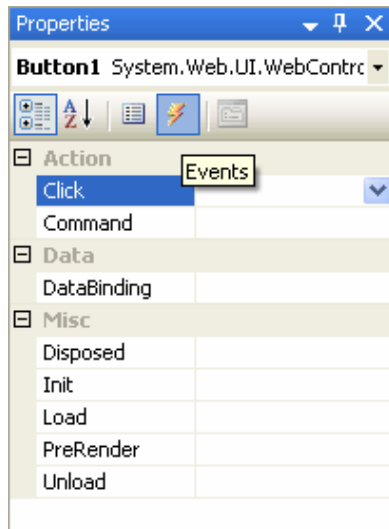
La idea de la programación orientada a eventos es que el programa realizará procesos o acciones según lo solicite o lo indique el usuario. Cuando haga clic en una casilla de verificación, arrastre un icono, cargue una página... en todas estos eventos colocaremos el código que queremos ejecutar.

Los antiguos programas realizaban una programación lineal, una instrucción detrás de otra, no existía una iteración con el usuario. Incluso cuando se añadieron las funciones y procedimientos no se realizaban cambios y la secuencia seguía siendo lineal.

El concepto de orientación a eventos cambia este panorama radicalmente a mejor. Windows por ejemplo no se ejecuta de forma lineal sino que está esperando eventos del usuario: escribir, hacer un clic con el ratón, pinchar una cámara, insertar un CD... en todos estos sucesos se activa un evento que hace que se ejecute el código necesario para atenderlo.

Normalmente el código de una página HTML se ejecuta sólo en el navegador y reacciona al ratón y poco mas, sin embargo con ASP.NET conseguiremos manejar los eventos desde el lado del servidor. Verás que esto es muy sencillo y potente ofreciéndonos unas posibilidades no vistas en ninguna otra plataforma.

Vamos a ver los eventos disponibles en los controles. Vete a la página donde hemos puesto los controles de ejemplo default.aspx y muestra la página web en la vista de diseño. Haz clic en el botón para seleccionarlo y luego mira a la derecha en la ventana de propiedades:



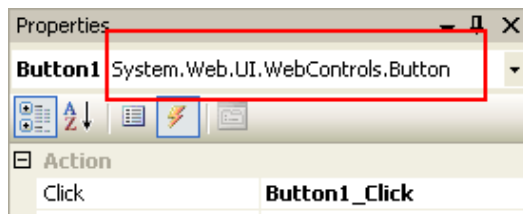
Selecciona el icono que representa un rayo, como ves en la pantalla anterior, aquí nos muestra los eventos del botón, es decir, las acciones que podremos realizar con él. La más lógica por supuesto es la del clic, así que suponemos que queremos escribir un código cuando se produzca este evento, es decir, cuando el usuario haga clic sobre él. Haz doble clic sobre esa línea (el evento click de la lista de eventos de la vista de propiedades) y automáticamente nos creará en nuestra página del código el "controlador del evento" es decir un procedimiento con el código que se ejecutará al hacer click:

```
Partial Class _Default
    Inherits System.Web.UI.Page

    Protected Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles Button1.Click

    End Sub
End Class
```

Antes de explicarte esto sólo un inciso, en la ventana de propiedades vemos lo siguiente:



Si te fijas nos está diciendo que el botón que hemos añadido a la página y que por defecto le ha llamado "button1" lo ha creado a partir de la clase "Button" que está en el espacio de nombres "System.Web.UI.WebControls". Sigamos ahora con el código que nos ha creado al señalar el evento click del botón.

Simplemente es un procedimiento (Sub) que se ejecutará cuando se haga click en el botón como indica la parte final de la declaración y mas importante:

Handles Button1.click

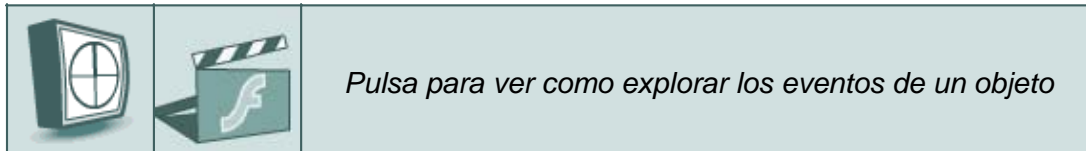
Este código del final es el que le indica que se ejecutará al hacer click. Si borramos este trozo del final ya no funcionará porque habremos borrado el controlador del evento clic del botón. Dentro de ese procedimiento

podremos poner el código que queramos, por ejemplo que escriba "Hola" en el cuadro de texto del ejemplo:

```
Protected Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles Button1.Click
```

```
    TextBox1.Text = "Hola"
```

```
End Sub
```



9.2 Intellisense y Outlinig

Estas dos técnicas son ayudas que nos ofrece el IDE para poder escribir el código de una forma mucho mas sencilla, ya que nos va a ayudar con la sintaxis dela escritura y con otros detalles que hacen de este IDE una herramienta verdaderamente potente.

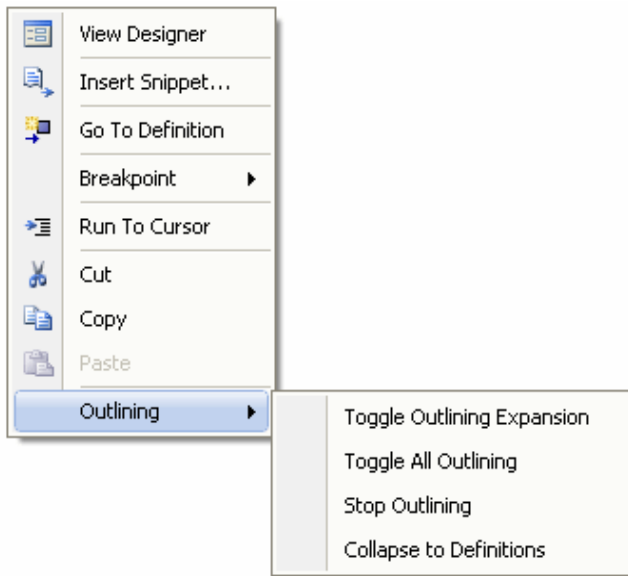
La primera que vemos es algo muy sencillo, el outlinig y es simplemente una mejora en el editor que nos permite ocultar o mostrar partes del código. Por ejemplo, tenemos nuestro ejemplo de antes así:

```
1
2 Partial Class _Default
3     Inherits System.Web.UI.Page
4
5     Protected Sub Button1_Click(ByVal sender As Obj
6         TextBox1.Text = "Hola"
7     End Sub
8 End Class
9
```

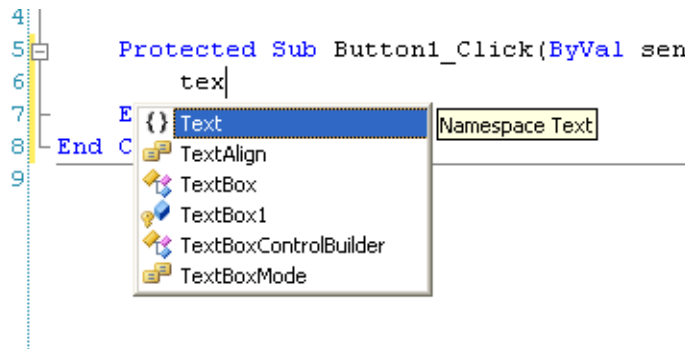
Fíjate que pasa si pulsamos en el signo "-" que he marcado en círculo rojo antes:

```
1
2 Partial Class _Default
3     Inherits System.Web.UI.Page
4
5     Protected Sub Button1_Click ...
6
7 End Class
8
9
```

Como ves simplemente nos amplía o encoje fragmentos de código. Desde el editor podremos expandir y contraer este código y así visualizar sólo la zona en la que estamos trabajando con lo que mejoraremos en claridad. Si pulsamos con el botón derecho en el editor tendremos todas las opciones de esta utilidad:

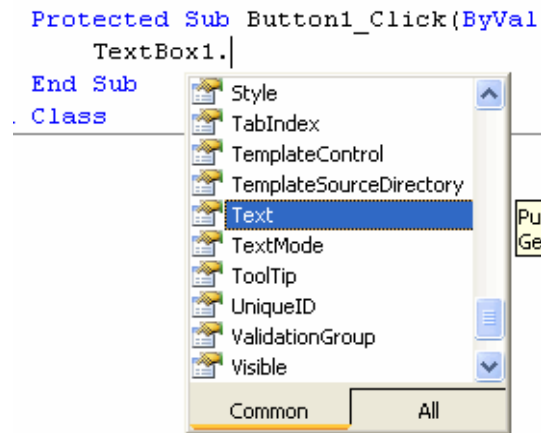


Ahora veremos el "intellisense" que si que es una verdadera utilidad para escribir nuestro código. Vamos a repetir la escritura de la asignación del cuadro de texto de antes, al comenzar a escribir el editor nos muestra lo siguiente:

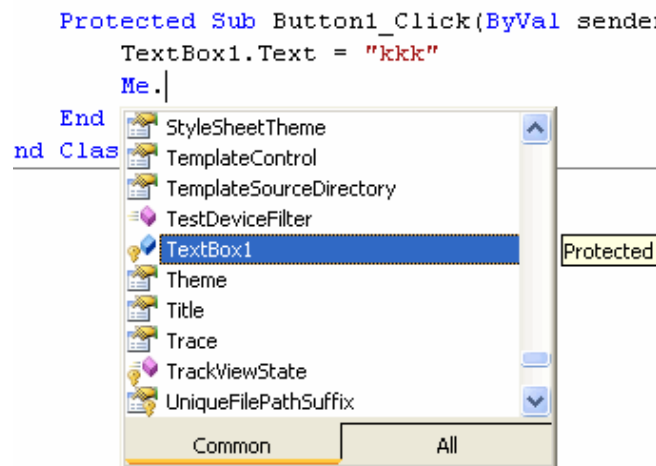


Al ir escribiendo letras el editor y gracias a esta tecnología nos muestra los objetos y operadores que podemos escribir, así, si seguimos escribiendo las letras del cuadro de texto, vemos que va apareciendo nuestro objeto "textbox1". Esto es una gran ayuda porque así no tengo que irme a ver el nombre del objeto, ya que con ir escribiendo sus letras iniciales nos irá indicando los objetos disponibles.

Pero mas interesante es cuando hemos escrito la palabra textbox y pulsamos el punto ".":



Porque nos va a mostrar todas las propiedades con las que podemos trabajar con este control u objeto. Mas ayudas, imagina que tenemos muchos controles en el formulario y no nos acordamos del nombre del alguno, basta con escribir "me." para que nos muestra todos los objetos del formulario actual (que le llama Yo o "me" en inglés):



Con solo escribir "me." el editor se ha posicionado en el primer control, que es precisamente con el que queremos trabajar. Además fíjate en el icono que le pone a la derecha, ese es el que indica que es un control, los que nos han salido en el ejemplo de antes que representa una mano seleccionando un texto es para las propiedades y tendremos otro icono en los eventos. Así que podremos seleccionar de la lista lo que necesitemos en ese momento.

Además de la ayuda en estas opciones también tenemos ayuda en la escritura de las instrucciones porque nos va a indicar las propiedades de las instrucciones que utilicemos. Por ejemplo, el método "Validate" realiza una operación que veremos mas adelante, este método puede necesitar algún parámetro, veamos que pasa al escribir:

Programación orientada a objetos. Formularios Web

```
5 Protected Sub Button1_Click(ByVal sender As  
6     TextBox1.Text = "kkk"  
7     Page.Validate(  
8         End 2 of 2 Validate (validationGroup As String)  
9 End Class validationGroup:  
10 The validation group name of the controls to validate.
```

Nos ha mostrado una ventana con "1 of 2", que quiere decir que está sobrecargado y se puede ejecutar sin parámetros o con un parámetro, como se muestra en la captura. Así que además de decirnos los parámetros que necesita y su tipo de datos, nos indica que puede estar sobrecargado y las variantes que tiene según los parámetros que le proporcionemos.

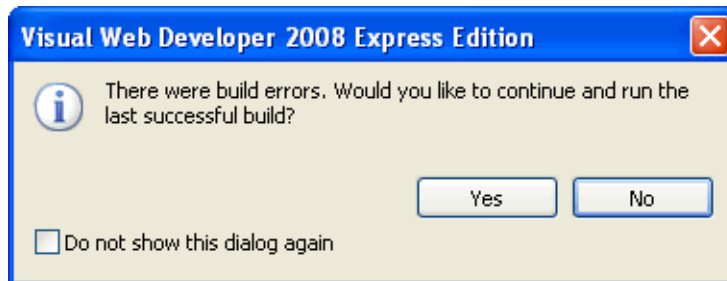
Mas cosas, cuando escribamos algo mal o nos falte algún dato o parámetro nos lo indicará con un subrayado en rojo:

```
Protected Sub Button1_Click(ByVal sender As  
    TextBox1.Te = "kkk"  
End Sub
```

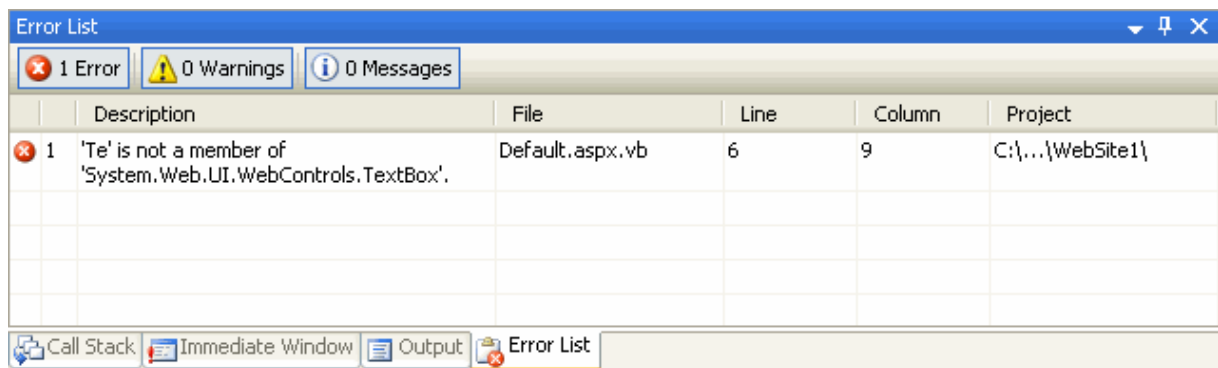
'Te' is not a member of 'System.Web.UI.WebControls.TextBox'.

En ese ejemplo, he escrito mal la propiedad "text", así que nos lo marca y si colocamos el ratón encima nos mostrará el error, en el ejemplo nos avisa que "Te" no es un miembro de ese objeto así que sabemos con toda seguridad que nos hemos equivocado ahí.

Aun así, si resulta que no lo hemos corregido o no nos hemos dado cuenta pasará lo siguiente al intentar ejecutar esta página con errores en el código:



Recuerda que comentamos que antes de ejecutar las páginas pasan por el proceso de traducción para convertirla en código ejecutable. En ese proceso se hacen las comprobaciones de la sintaxis y nos avisa con esa pantalla, damos a "Yes" para que nos muestre los detalles:



Programación orientada a objetos. Formularios Web

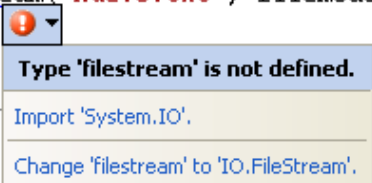
Con esto aprendemos una ventana mas: la de la lista de errores. Ahí nos mostrará con detalle todos los errores del código de la página. En ocasiones no serán "Error" sino que serán "Warnings" o advertencias. En ese caso se puede producir potencialmente un error y nos lo avisa.

La técnica de "Intellisense" no solo nos ayuda en la sintaxis sino que nos puede sugerir correcciones con una utilidad llamada "AutoCorrect", que obviamente significa autocorrección. Vamos a escribir esta línea de código:

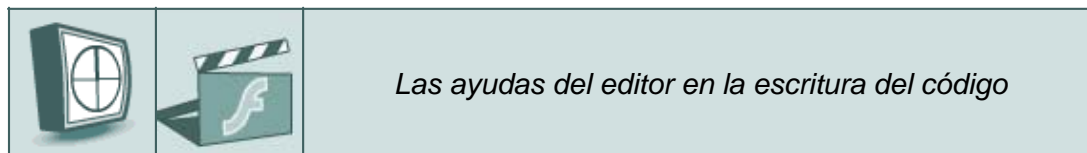
```
Protected Sub ButtType 'filestream' is not defined. der As Object, ByVal e As  
    Dim fs As New filestream("Nuevo.txt", FileMode.Create)  
End Sub
```

Estamos creando un objeto de la clase "filestream", al escribirla ya no nos aparecía ayuda sobre ella, esa es una buena pista para saber que no lo estamos escribiendo bien, pero la terminamos de escribir y tenemos un error que lo subraya en azul. El mensaje de error es muy ambiguo ya que solo me dice que esa clase no está definida. Veamos como nos ayuda a resolver el error, pulsamos en la admiración y tenemos:

```
Protected Sub Button1_Click(ByVal sender As Object, ByVal e  
    Dim fs As New filestream("Nuevo.txt", FileMode.Create)  
End Sub  
End Class
```



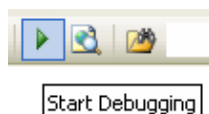
Muy bien, nos avisa que "filestream" pertenece al espacio de nombres "System.IO" así que nos muestra dos soluciones posibles: una importar a nuestra página el espacio de nombres adecuado o poner el nombre completo de la clase: "IO.FileStream". Lo veremos en acción muchas veces porque en nuestro aprendizaje nos equivocaremos unas cuantas veces.



9.3 Depuración

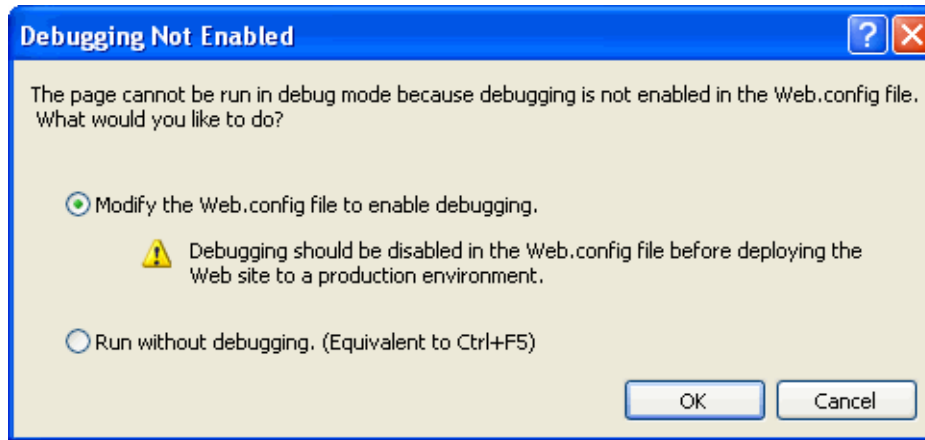
La depuración es una forma de ejecutar las páginas de forma controlada para que podamos ver la ejecución paso a paso y podamos analizar la ejecución. Es una de las herramientas mas potentes en nuestro IDE y un gran avance que nos va a facilitar la localización y corrección de errores.

La ejecución en modo de depuración ya lo conoces porque lo hemos utilizado varias veces y es el botón de inicio de la barra de herramientas:



Programación orientada a objetos. Formularios Web

Para que en nuestro sitio web podamos ejecutar estas depuraciones debemos poner una instrucción en el fichero de configuración del web: "web.config". Por suerte el IDE también se cuenta de que no está inicialmente puesto y nos propone ponerlo por nosotros, esta pantalla ya la habíamos visto antes pero ahora ya sabemos el porqué. Luego al intentar ejecutar un web en modo de depuración y que no tenga la activación puesta en el fichero de configuración, obtendremos este mensaje:



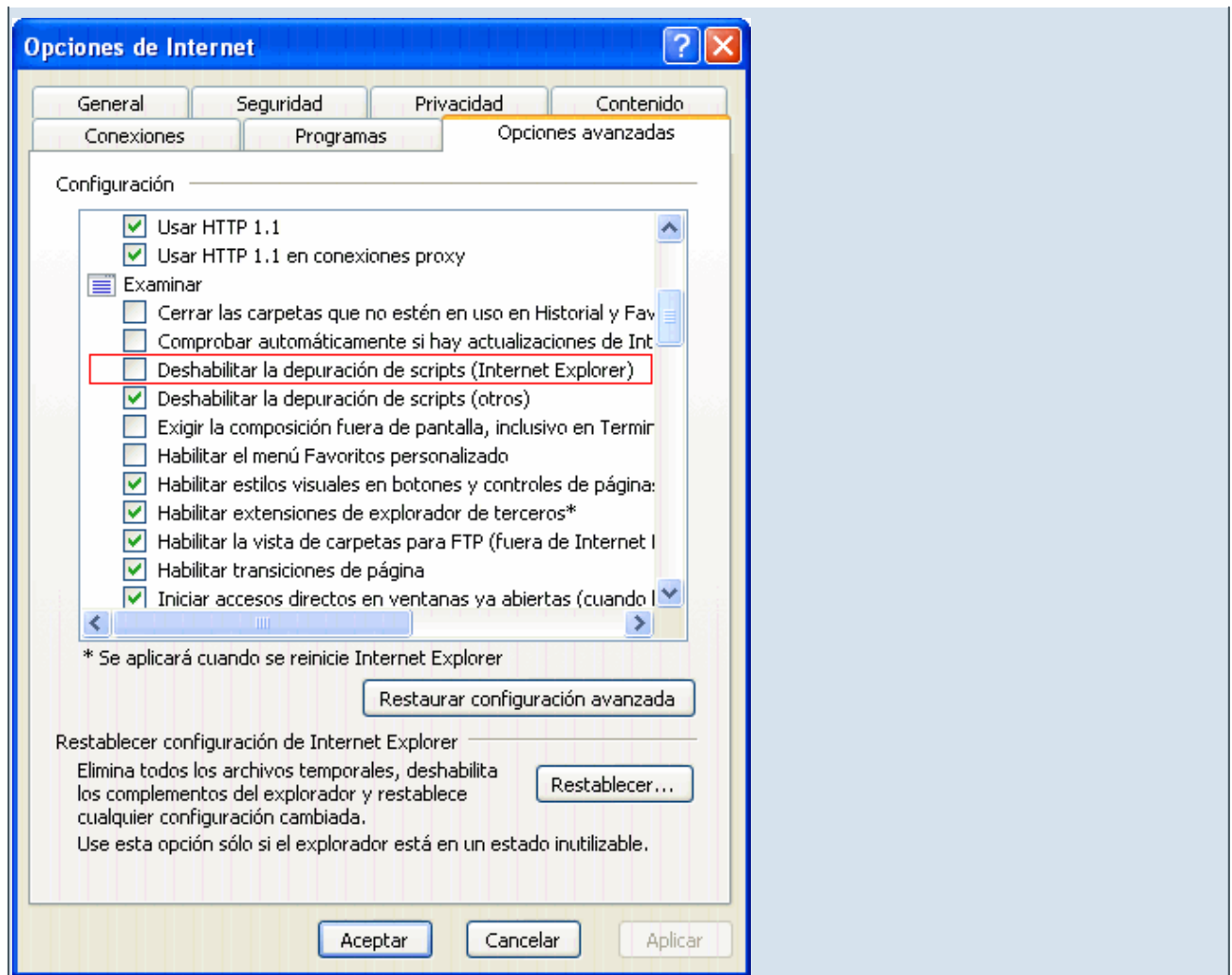
Le diremos que nos modifique el fichero y continuaremos. En realidad simplemente ha cambiado esta opción en el fichero de configuración:

```
<system.web>
  <!--
    Set compilation debug="true" to insert debugging
    symbols into the compiled page. Because this
    affects performance, set this value to true only
    during development.

    Visual Basic options:
    Set strict="true" to disallow all data type conversions
    where data loss can occur.
    Set explicit="true" to force declaration of all variables.
  -->
  <compilation debug="true" strict="false" explicit="true">
```

El comentario en verde nos indica que la depuración afecta al rendimiento ya que incluye mucha información adicional para ayudarnos a solucionar y realizar seguimiento al código. Así que solo lo utilizaremos ahora que estamos desarrollando las páginas. Para desactivarlo simplemente pondríamos a "false" la opción "debug" de la parte de debajo.

Nota: Para que en nuestro navegador nos muestre toda la información de depuración de la página debemos desactivar lo siguiente en el Internet Explorer y dentro de las Opciones del menú de Herramientas:



9.4 El servidor Web

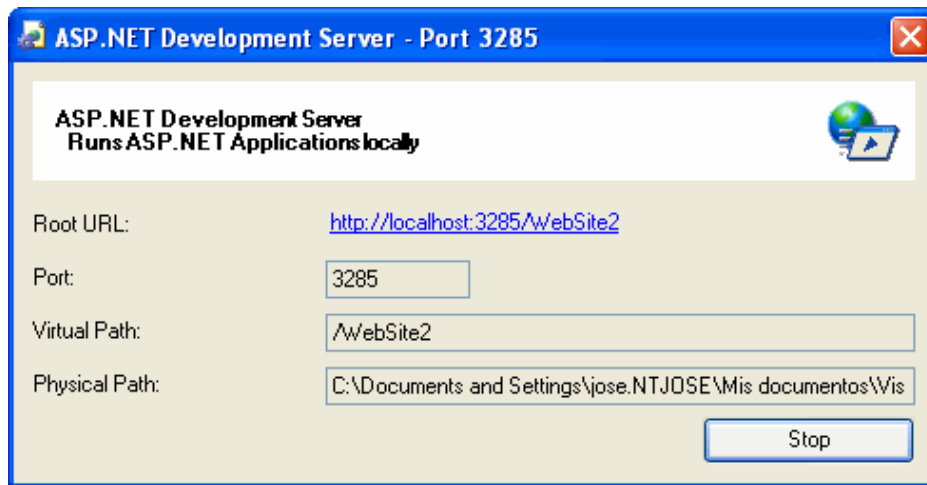
Como último elemento de ayuda del entorno para poder desarrollar las páginas finalizamos este tema. Ya hemos visto que al ejecutarse las páginas se pone en marcha un servidor web local, herramienta imprescindible porque las páginas se ejecutan siempre bajo un servidor web.

En condiciones normales utilizaremos el IIS de Windows 2003 ó 2008 Server pero para las pruebas y desarrollo podemos utilizar este servidor local. Veamos que cosas pasaban al ejecutar la página en modo de depuración, por un lado el servidor web en ejecución en los iconos de la barra de tareas:

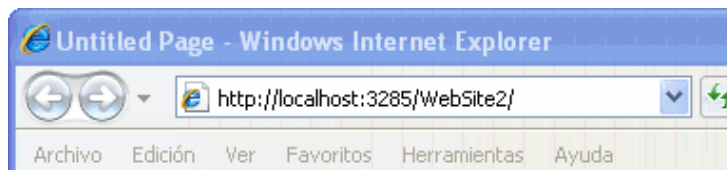
ASP.NET Development Server - Port 3285



Si hacemos doble clic sobre el icono, veremos más información sobre este servidor:



Nos dice en que directorio físico corresponde al sitio web que está ejecutando. Recuerda los directorios virtuales del tema 2. También nos dice que se ejecuta en el puerto 3285, pero es un valor que no nos importa porque son ejecuciones temporales para depuración. Lógicamente en la vista de la página en el navegador, le indica este puerto y el directorio virtual del sitio web:



"Localhost" es una palabra clave que le indica que la página está en un servidor del propio ordenador y que el servidor web está en el puerto 3285. Finalmente muestra la página predeterminada (recuerda según el tema 2: default.aspx) del sitio virtual "website2" que es precisamente el que hemos creado desde el IDE. Por tanto al darle a la depuración crea un directorio virtual en el web que hemos creado en el IDE y nos lo muestra en el web.

9.5 Una depuración sencilla

Hemos puesto en marcha nuestra página en modo de depuración pero no hemos hecho uso de ellas, así que vamos a realizar un sencillo ejemplo. Tenemos una página con un botón en un formulario. Hacemos doble clic en él y nos creará automáticamente el controlador de este evento (otra forma de crearlo mas práctica que desde la ventana de propiedades que vimos antes). Escribimos este código que aunque no realice nada en nuestra página nos va a servir para ilustrar cómo depurar una página.

Queremos ver que pasa cuando se ejecute la instrucción que realiza la suma ya que sospechamos que ahí no funciona algo bien: pondremos un punto de interrupción haciendo clic en la parte de la izquierda de la línea fuera del editor hasta que nos la marque con un círculo:

```

1
2 Partial Class _Default
3     Inherits System.Web.UI.Page
4
5     Protected Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
6         Dim resultado As Long
7         Dim sum1, sum2 As Long
8
9         sum1 = 20
10        sum2 = 30
11
12        resultado=sum1+ sum1
13
14    End Sub
15 End Class
16

```

Un punto de interrupción es un punto donde el programa va a detener la ejecución para permitirnos ver el contenido de sus variables y poder ejecutar las instrucciones una a una para poder realizar un seguimiento de su funcionamiento. Ponemos en marcha la página desde el botón de iniciar depuración y se nos mostrará la página web resultante en el navegador. Pero al hacer clic en el botón la ejecución se irá al controlador del evento clic que hemos creado y se detendrá en el punto de interrupción, marcando con amarillo que la ejecución se encuentra detenida en esa línea:

```

1
2 Partial Class _Default
3     Inherits System.Web.UI.Page
4
5     Protected Sub Button1_Click(ByVal sender As Object, B
6         Dim resultado As Long
7         Dim sum1, sum2 As Long
8
9         sum1 = 20
10        sum2 = 30
11
12        resultado = sum1 + sum1
13
14    End Sub
15 End Class

```

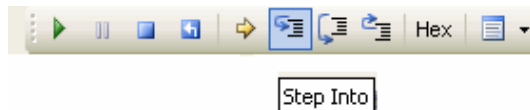
La información que nos va a proporcionar es enorme, nos fijamos en la ventana inferior:

Name	Value	Type
Me	{ASP.default_aspx}	_Default
e	{System.EventArgs}	System.E
resultado	0	Long
sender	{Text = "Button"}	Object
sum1	20	Long
sum2	30	Long

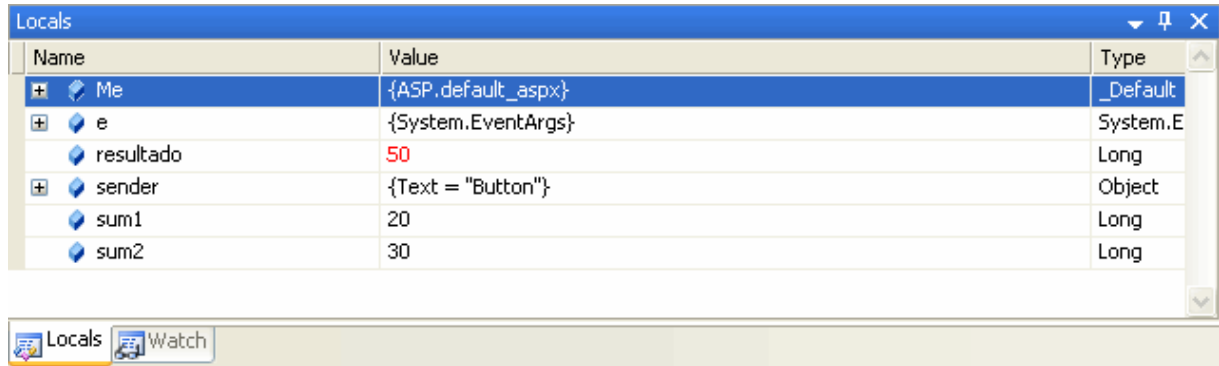
Donde podemos ver el valor de todas las variables y objetos de nuestra página, vemos por ejemplo como "sum1" y "sum2" tienen los valores 20 y 30 y que en esa línea "resultado" todavía tiene el valor 0. Seguiremos

Programación orientada a objetos. Formularios Web

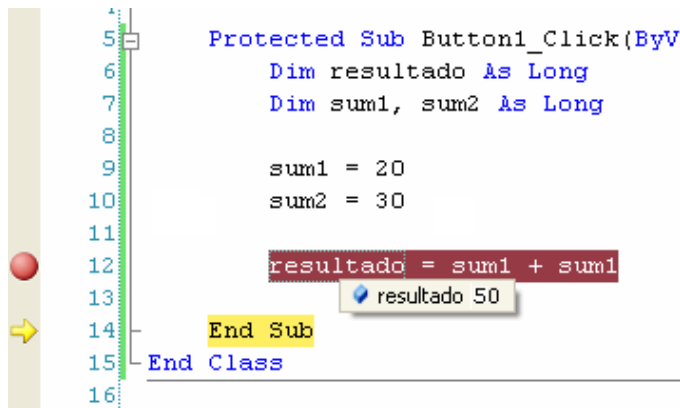
con la ejecución línea a línea pulsando en el icono de "Step Into":



Al pulsar una vez hemos forzado la ejecución de esa línea, hará la operación y habremos visto que la variable se ha actualizado, marcándolo en rojo:



Además poniendo el cursor encima de cualquier variable nos dirá su valor actual:

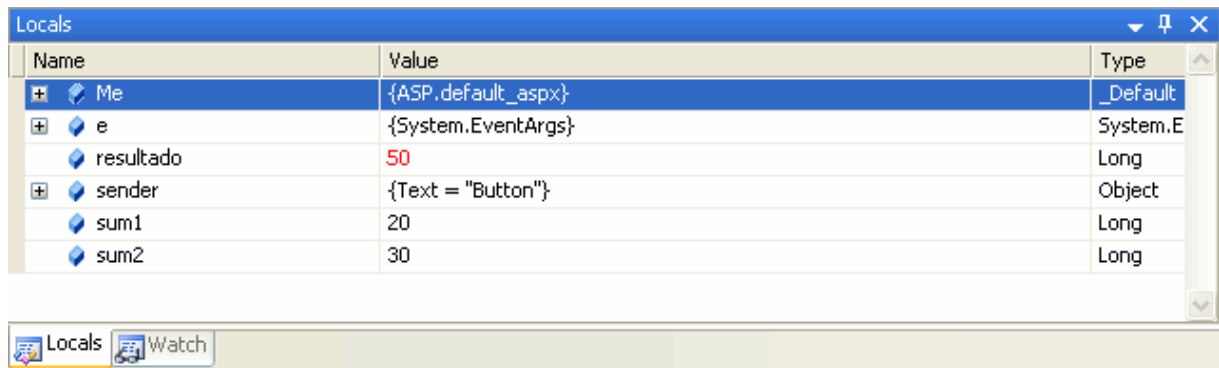


Ya ves que las posibilidades son muchas y la depuración se convertirá a partir de ahora en la principal herramienta de desarrollo porque podremos depurar y seguir la ejecución de todas nuestras páginas con toda seguridad. Todas las opciones necesarias las tienes en el menú "Debug". Aquí tienes todas descritas y cómo se ejecutan:

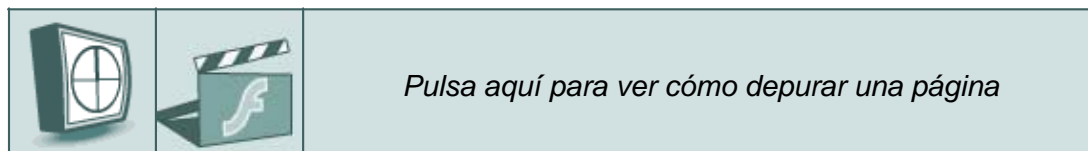
Comando	Descripción
Step Into F8	Ejecuta la línea actual y se detiene. Es decir hace una ejecución línea a línea.
Setp Over (Mayús + F8)	Como la anterior pero no se mueve a los procedimientos. Es decir si hay una instrucción que llama a un procedimiento, lo ejecuta sin meterse dentro de él.
Setp Out (Ctrl + mayús + F8)	Ejecuta todo el código del procedimiento actual y se detiene al terminarlo para seguir depurando
Continue (F5)	Continúa la ejecución normal de la página, sin detenerse mas
Run to Cursor	Ejecuta el programa hasta donde está situado el cursor, en ese momento entra en depuración.
Set Next Statement	Permite cambiar la ruta de ejecución del programa durante la depuración.
Show Next Statement	Muestra la siguiente línea que se ejecutará.

Programación orientada a objetos. Formularios Web

Y ya para finalizar recordarte la ventana de visualización de variables que vimos antes:



Donde, sin poner puntos de ruptura, podemos ver el estado de las variables.

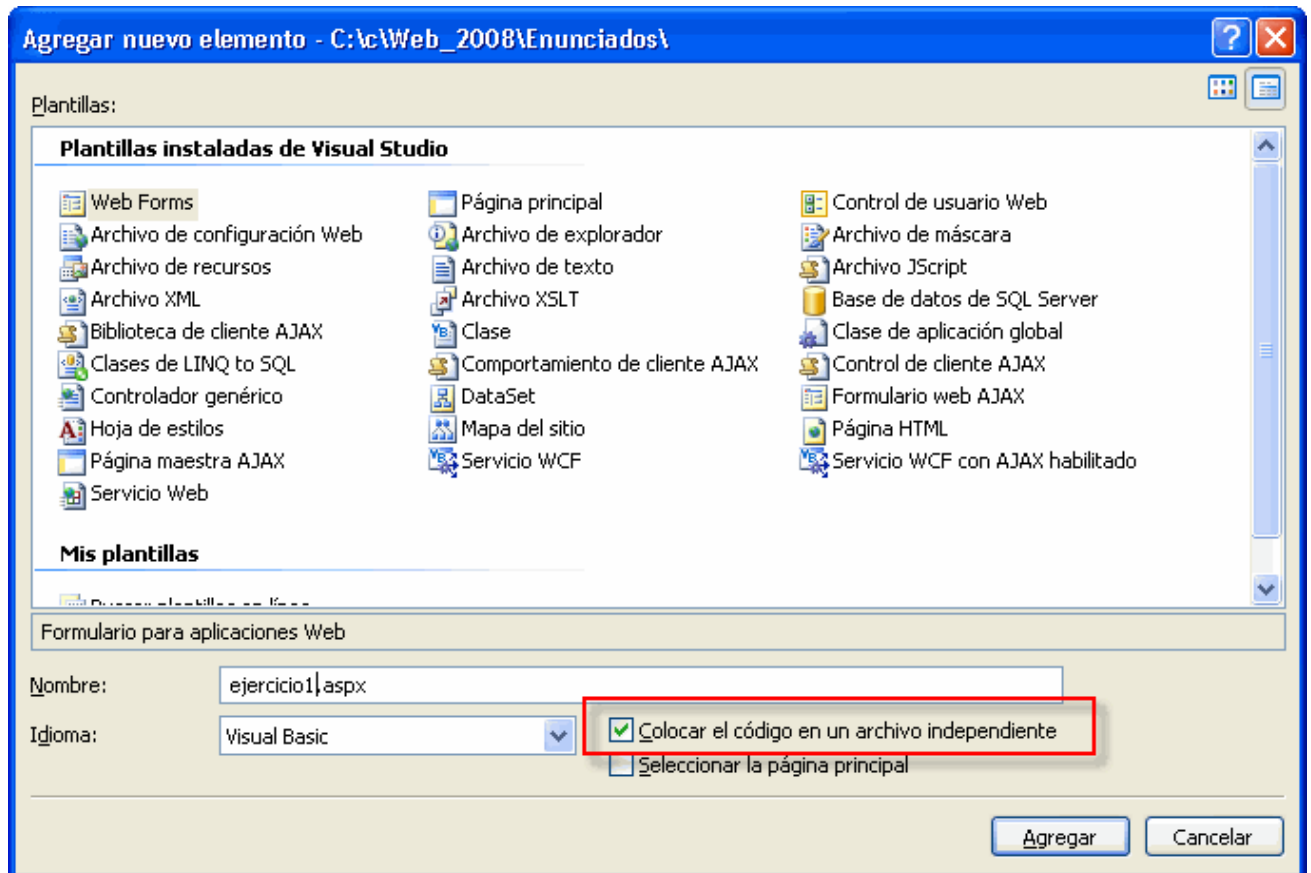


[Pulsa aquí para descargar los ejemplos de este tema](#)

Ejercicios

Ejercicio 1

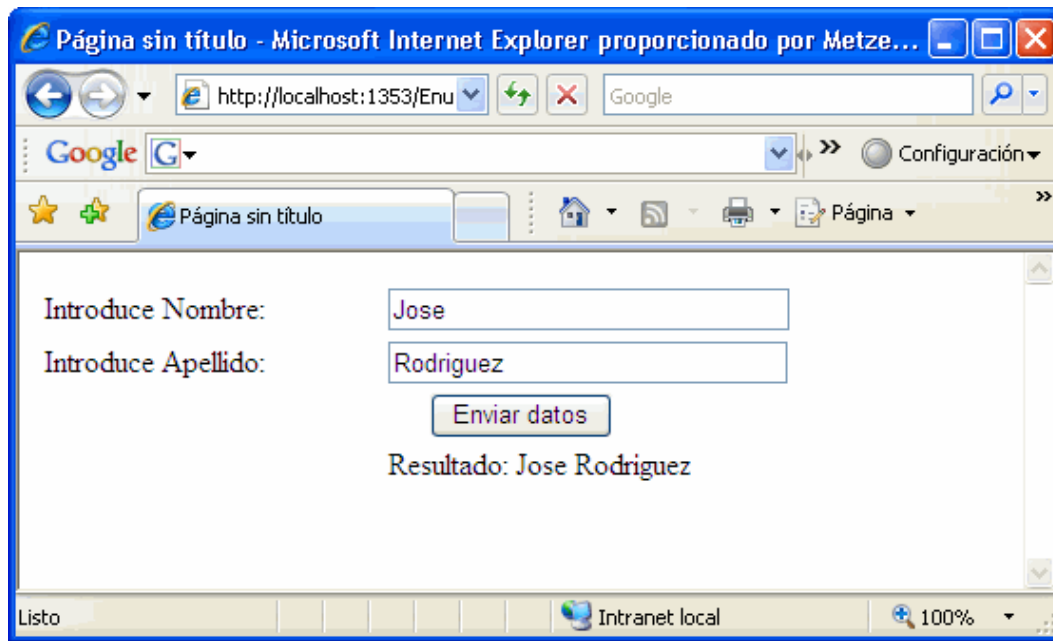
Empezaremos a trabajar con los eventos de Windows, así que según el vídeo de ejemplo donde se explicaban los pasos para recoger el evento clic de un botón. A partir de ahora siempre utilizaremos la página de código independiente para separar el código HTML del ASP.NET:



Crea una página con dos cuadros de texto, una etiqueta Label y un botón. Ayúdate de las tablas para colocar los controles en pantalla. Cuando se haga clic debemos escribir el resultado en las etiquetas "Label":

Introduce Nombre:	<input type="text"/>
Introduce Apellido:	<input type="text"/>
<input type="button" value="Enviar datos"/>	
Resultado: [lb_resultado]	

Buscaremos el evento clic para asignar a la propiedad "Text" de la etiqueta el contenido de los cuadros de texto (propiedad Text). Para que de como resultado:



Ejercicio 2

Con la misma filosofía que el anterior, crea una sencilla calculadora. Debes declarar las variables de tipo entero y convertir los datos de los cuadros de texto a numérico para poder hacer la operación. Escribe en el evento de cada botón la operación a realizar, no importa que repitas líneas de código:

Primer operando:	<input type="text"/>
Segundo operando:	<input type="text"/>
<div><input type="button" value="+"/> <input type="button" value="-"/> <input type="button" value="x"/> <input type="button" value="/"/></div>	
El resultado es: [lb_resultado]	

El resultado será:

The screenshot shows a Microsoft Internet Explorer browser window. The title bar reads "Página sin título - Microsoft Internet Explorer proporcionado por Metzeler ...". The address bar shows the URL "http://localhost:1353/Enunciac". The search bar contains the text "Google". The browser's toolbar includes buttons for back, forward, home, and search, as well as a "Configuración" (Settings) button. The main content area displays a web form with the following elements:

- Two input fields for operands. The first is labeled "Primer operando:" and contains the value "4". The second is labeled "Segundo operando:" and contains the value "9".
- Four buttons for arithmetic operations: addition (+), subtraction (-), multiplication (x), and division (/).
- A text label "El resultado es: 13" indicating the result of the calculation.

The status bar at the bottom shows "Intranet local" and a zoom level of "100%".