Properties - The complete C# tutorial

9-11 minutos

Clases:

En el artículo anterior, vimos los **campos**. Son como las variables globales de una clase, y permiten acceder a ellos desde todos los métodos. También vimos que, de hecho, los campos PUEDEN accederse desde otras clases si se los define como **públicos**, pero eso, en general, no se recomienda. Para las variables o campos que se quieran acceder desde fuera de la clase, en cambio se deberían usar **propiedades**.

Cuando usted declara un campo como público, está dando acceso completo a el desde afuera - otras clases pueden hacer lo que quieran con él, sin notificarlo la clase declarante. Las propiedades regresan el control a la clase declarante, al especificar si un campo es de solo lectura o escritura y aún permitiendo a la clase declarante verificar y manipular el valor antes de retornarlo o asignarlo al campo.

Una propiedad se ve como una mezcla entre un campo y un método, ya que es declarado mas como un campo con visibilidad, un tipo de dato y un nombre, pero tiene también un cuerpo como un método, para controlar el comportamiento:

```
public string Name
{
        get { return _name; }
        set { _name = value; }
}
```

Note las palabras clave especiales **get** y **set**. Ellas son usadas exclusivamente para propiedades, para controlar el comportamiento cuando el campo es leido (get) y escrito (set). Usted puede declarar propiedades con sólo una implementación de get O de un set, por ejemplo, para crear una propiedad la cual pueda ser leida desde cualquier parte (pública) pero sólo ser modificada desde dentro de la clase declarante (privada).

También notará que me refiero a un campo llamado _name. Usted tendrá que declarar este en la clase también, de tal forma que su propiedad pueda usarla. Un patrón de uso común para campos y propiedades se verá como este:

```
private string _name = "John Doe";

public string Name
{
        get { return _name; }
        set { _name = value; }
}
```

Ahora puede ver como el campo y la propiedad trabajan juntos: El **método get** regresará el valor del campo **_name**, mientras que el **método set** asignará el valor pasado al campo **_name**. En el método set, usamos la palabra clave especial value la cual, en esta situación específica, referirá al valor pasado a la propiedad.

Así, esto es prácticamente todo lo básico como puede ser y en este punto, no hicimos nada que no pueda ser obtenido con un simple campo público. Pero en un punto posterior, usted pudiera decidir que quiere tomar un mayor control de como otras clases puede trabajar con el nombre y ya que usted ha implementado esto como un propiedad, es libre de modificar la implementación sin disturbios para quien use su clase. Por ejemplo, la propiedad Name puede ser modificada para verse como esto:

```
private string _name = "John Doe";

public string Name
{
        get
        {
                return _name.ToUpper();
        }
        set
        {
                if(!value.Contains(" "))
                        throw new Exception("Please specify both first and last
name!");
                _name = value;
        }
}
```

El **método get** ahora forza a que el valor retornado sea siempre en MAYÚSCULAS, no importa en que tipo esté el campo de respaldo (_name). En el **método set**, hemos agregado un par de líneas de código para verificar si el valor pasado contiene un espacio en blanco, porque hemos decidido que el nombre siempre debe consistir de ambos un nombre y un apellido - si este no es el caso, una excepción es lanzada. Esto es todo muy crudo y simplificado, pero debe ilustrar el nivel completo de control que usted obtiene cuando usa propiedades.

Propiedades Solo lectura

La mayoría de las propiedades que verá en los ejemplos de este tutorial serán tanto de lectura como de escritura, porque ese es el uso más común de las propiedades, pero no siempre es así. En primer lugar, puede declarar una propiedad usando solo el método get, por ejemplo:

```
private string _name = "John Doe";

public string Name
{
        get { return _name; }
}
```

En este caso, ya no puede cambiar la propiedad "Name": solo puede leerlo y el compilador arrojará un error si intenta asignarle un valor. Sin embargo, aún puede cambiar su valor desde dentro de la clase, ya que simplemente puede asignar un nuevo valor al campo de respaldo "_name". Sin embargo, hacerlo de esa manera niega una de las mayores ventajas de las propiedades: la capacidad de controlar siempre si se puede aceptar un valor. Como ya mencionamos, el método set es una excelente manera de realizar la validación del valor, excepto si asigna un nuevo valor al campo _name desde varios lugares, debido a que la propiedad es de solo lectura, no obtiene esta validación

Afortunadamente para nosotros, C# ofrece una solución a esto: puede definir un método set en la propiedad, pero limitar su visibilidad, usando, por ejemplo la palabra clave **private** o **protected**.

Esto le dará lo mejor de ambos mundos, donde aún puede asignar un valor a la propiedad desde dentro de la clase (o cualquier clase heredada si usa la palabra clave protected) y validarlo en consecuencia. Aquí hay un ejemplo:

```
private string _name = "John Doe";

public string Name
{
        get { return _name; }

        private set
        {
                if(IsValidName(value))
                        this._name = value;
        }
}

public bool IsValidName(string name)
{
        return name.EndsWith("Doe");

}
```

La diferencia clave aquí es simplemente la palabra clave "privada" justo en frente de la palabra clave "set" y, como se mencionó, puede reemplazarla con, por ejemplo, protected o internal, según sus necesidades.

Propiedades Auto-implementadas

En algunos casos, no necesita todo el control sobre los campos y se puede sentir incómodo de implementar ambos un campo y un propiedad con los métodos get y set sin hacer nada adicional a lo que vimos en el primer ejemplo. Usted puede ser tentado a declarar simplemente sus variables como un campo público y evitar toda esta molestia. Pero no lo haga! Afortunadamente para todos nosotros, Microsoft decidió agregar las propiedades auto-implementadas en C# versión 3, lo cual le ahorrará varias líneas de código. Sólo considere la diferencia:

Propiedad regular con un campo de respaldo declarado:

```
private string _name;

public string Name
{
        get { return _name; }
        set { _name = value; }
}
```

El mismo comportamiento exacto, pero con una propiedad auto-implementada:

```
public string Name { get; set; }
```

Note que los métodos get y set están vacíos y que no se declara un campo de respaldo - en otras palabras, ¡Ahora podemos obtener el mismo comportamiento exacto como en el primer ejemplo pero con un sola línea de código! Tenga en mente que un campo de respaldo privado aún existirá en el tiempo de ejecución - será auto-implementado por el compilador, como lo implica su nombre. Más tarde podría decidir que usted necesita más control de esta propiedad específica, usted puede

simplemente cambiarla a una combinación regular de campo/propiedad con la implementación deseada de los métodos get y set.

Note que aún tiene un mecanismo importante de control de las propiedades regulares cuando usa propiedades auto-implementadas: Usted puede dejar fuera las palabra clave set para crear un propiedad de sólo lectura, por ejemplo como este:

```
public string ReadOnlyProperty { get; }
```

Las propiedades de sólo escritura no son permitidas cuando se usan propiedades auto-implementadas.

Propiedades auto-implementadas con valores default

Previo a C# versión 6, usted no podía definir un valor default (por omisión) de una propiedad auto-implementada - para ello, usted necesitaría un campo de respaldo declarado, el cual le permitiría inicializar la variable con un valor:

```
private string _name = "John Doe";

public string Name
{
        get { return _name; }
        set { _name = value; }
}
```

Pero en C# versión 6, Microsoft finalmente agregó la abilidad para inicializar una propiedad auto-implementada con un valor default, como esta:

```
public string Name { get; set; } = "John Doe";
```

Propiedades con cuerpo de expresión

Otra característica relacionada con las propiedades que Microsoft implementó en C# 6.0 y 7.0 son los miembros con cuerpo de expresión. Se permite simplemente escribir expresiones de una sola línea para las propiedades y los métodos - en este caso, veamos cómo usarlo para tus métodos get/set de manera que tome menos espacio y requiera escribir un poco menos.

```
private string name;
public string Name
{
        get => name;
        set => name = value;
}
```

Si la propiedad es de solo lectura, la sintaxis es todavía mas corta:

```
public string Name => "John Doe";
```

Por supuesto esto también funciona si necesitas hacer algo antes de retornar el valor, como ahora:

```
public string Name { get; set; } = "John Doe";

public string FirstName => this.Name.Substring(0, this.Name.IndexOf(" "));
```

Como puedes ver, esto te permite definir un metodo get, pero sin las palabras *get* y *return*, mientras que anima a mantener todo en una sola línea en lugar de usar varias.

Resumen

Las propiedades le dan a su clase más control sobre como los campos puedes ser accedidos y manipulados, y estas siempre deben ser usadas cuando usted quiere dar acceso a los campos desde fuera de la clase declarante.