Programación orientada a objetos (C# y Visual Basic)

**Visual Studio 2012**

Otras versiones

Personas que lo han encontrado útil: 7 de 7 - [Valorar este tema](http://msdn.microsoft.com/es-es/library/dd460654.aspx#feedback)

Todos los lenguajes administrados en el.NET Framework, como Visual Basic y C#, proporcionan compatibilidad completa para la programación orientada a objetos como encapsulación, herencia y polimorfismo.

*Encapsulación* significa que un grupo de propiedades relacionadas, métodos y otros miembros se tratan como una sola unidad u objeto.

*Herencia* Describe la capacidad de crear nuevas clases basadas en una clase existente.

*Polimorfismo* significa que puede tener varias clases que se pueden utilizar indistintamente, aunque cada clase implementa las mismas propiedades o métodos de maneras diferentes.

En esta sección se describe los siguientes conceptos:

* [Clases y objetos](http://msdn.microsoft.com/es-es/library/dd460654.aspx#Classes)
	+ [Miembros de clase](http://msdn.microsoft.com/es-es/library/dd460654.aspx#Members)

[Los campos y propiedades](http://msdn.microsoft.com/es-es/library/dd460654.aspx#Properties)

[Métodos](http://msdn.microsoft.com/es-es/library/dd460654.aspx#Methods)

[Constructores](http://msdn.microsoft.com/es-es/library/dd460654.aspx#Constructors)

[Destructores](http://msdn.microsoft.com/es-es/library/dd460654.aspx#Destructors)

[Eventos](http://msdn.microsoft.com/es-es/library/dd460654.aspx#Events)

[Clases anidadas](http://msdn.microsoft.com/es-es/library/dd460654.aspx#NestedClasses)

* + [Modificadores de acceso y los niveles de acceso](http://msdn.microsoft.com/es-es/library/dd460654.aspx#AccessModifiers)
	+ [Instancias de clases](http://msdn.microsoft.com/es-es/library/dd460654.aspx#InstantiatingClasses)
	+ [Estático (compartido) clases y miembros](http://msdn.microsoft.com/es-es/library/dd460654.aspx#Static)
	+ [Tipos anónimos](http://msdn.microsoft.com/es-es/library/dd460654.aspx#AnonymousTypes)
* [Herencia](http://msdn.microsoft.com/es-es/library/dd460654.aspx#Inheritance)
	+ [Miembros de reemplazo](http://msdn.microsoft.com/es-es/library/dd460654.aspx#Overriding)
* [Interfaces](http://msdn.microsoft.com/es-es/library/dd460654.aspx#Interfaces)
* [Componentes genéricos](http://msdn.microsoft.com/es-es/library/dd460654.aspx#Generics)
* [Delegados](http://msdn.microsoft.com/es-es/library/dd460654.aspx#Delegates)

Clases y objetos

Los términos *clase* y *objeto* se utilizan a veces indistintamente, pero de hecho, las clases describen la *tipo* de objetos, mientras que los objetos son utilizables *instancias* de clases. Por lo tanto, se llama la acción de crear un objeto *creación de instancias*. Con la analogía de plano, una clase es un plano y un objeto es un edificio construido a partir de ese plano.

Para definir una clase:

C#

[**VB**](http://msdn.microsoft.com/es-es/library/dd460654.aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1)

class SampleClass

{

}

Visual Basic y C# también proporcionan una versión "light" de clases llamadas *estructuras* que son útiles cuando se necesita crear una matriz grande de objetos y hacer consumir demasiada memoria para el que desea.

Para definir una estructura:

C#

[**VB**](http://msdn.microsoft.com/es-es/library/dd460654.aspx?cs-save-lang=1&cs-lang=vb#code-snippet-2)

struct SampleStruct

{

}

Para obtener más información, consulte:

* Visual Basic
	+ [Instrucción Class (Visual Basic)](http://msdn.microsoft.com/es-es/library/wa0hwf23.aspx)
	+ [Structure (Instrucción)](http://msdn.microsoft.com/es-es/library/k69kzbs1.aspx)
* C#
	+ [class (Referencia de C#)](http://msdn.microsoft.com/es-es/library/0b0thckt.aspx)
	+ [struct (Referencia de C#)](http://msdn.microsoft.com/es-es/library/ah19swz4.aspx)

Miembros de clase

Cada clase puede tener diferentes *miembros de clase* que incluyen propiedades que describen los datos de la clase, los métodos que definen el comportamiento de la clase y los eventos que proporcionan comunicación entre las distintas clases y objetos.

Los campos y propiedades

Campos y propiedades representan información que contiene un objeto. Los campos son igual que las variables ya que se pueden leer o establecer directamente.

Para definir un campo:

C#

[**VB**](http://msdn.microsoft.com/es-es/library/dd460654.aspx?cs-save-lang=1&cs-lang=vb#code-snippet-3)

Class SampleClass

{

 public string sampleField;

}

Propiedades tienen get y establecer los procedimientos, que proporcionan más control sobre cómo se establecen o devuelven los valores.

C# y de Visual Basic permiten ya sea crear un campo privado para almacenar el valor de propiedad o llamadas propiedades implementadas automáticamente que crean este campo automáticamente entre bastidores y proporcionan la lógica básica para los procedimientos de propiedad.

Para definir una propiedad implementada automáticamente:

C#

[**VB**](http://msdn.microsoft.com/es-es/library/dd460654.aspx?cs-save-lang=1&cs-lang=vb#code-snippet-4)

class SampleClass

{

 public int SampleProperty { get; set; }

}

Si necesita realizar algunas operaciones adicionales para leer y escribir el valor de propiedad, defina un campo para almacenar el valor de propiedad y proporcionar la lógica básica para almacenar y recuperarlo:

C#

[**VB**](http://msdn.microsoft.com/es-es/library/dd460654.aspx?cs-save-lang=1&cs-lang=vb#code-snippet-5)

class SampleClass

{

 private int \_sample;

 public int Sample

 {

 // Return the value stored in a field.

 get { return \_sample; }

 // Store the value in the field.

 set { \_sample = value; }

 }

}

Mayoría de las propiedades tiene métodos o procedimientos para configurar y obtener el valor de propiedad. Sin embargo, puede crear propiedades de sólo lectura o sólo escritura para restringir su modificación o la lectura. En Visual Basic se puede utilizar **Sólo lectura** y **WriteOnly** palabras clave. En C#, se puede omitir el **Get** o **conjunto** método de propiedad. Sin embargo, en Visual Basic y C#, las propiedades implementadas automáticamente no pueden ser de sólo lectura o sólo escritura.

Para obtener más información, consulte:

* Visual Basic
	+ [Property (Instrucción)](http://msdn.microsoft.com/es-es/library/zzh9ha57.aspx)
	+ [Get (Instrucción)](http://msdn.microsoft.com/es-es/library/38x6w70d.aspx)
	+ [Instrucción Set (Visual Basic)](http://msdn.microsoft.com/es-es/library/6b85bc00.aspx)
	+ [ReadOnly (Visual Basic)](http://msdn.microsoft.com/es-es/library/z2b2c2ka.aspx)
	+ [WriteOnly (Visual Basic)](http://msdn.microsoft.com/es-es/library/25dey2kc.aspx)
* C#
	+ [get (Referencia de C#)](http://msdn.microsoft.com/es-es/library/ms228503.aspx)
	+ [set (Referencia de C#)](http://msdn.microsoft.com/es-es/library/ms228368.aspx)

Métodos

A *método* es una acción que puede realizar un objeto.

|  |
| --- |
| **NotaNota** |
| En Visual Basic, hay dos formas de crear un método: el **Sub** se utiliza la instrucción si el método no devuelve un valor; el **Función** se utiliza la instrucción si un método devuelve un valor. |

Para definir un método de una clase:

C#

[**VB**](http://msdn.microsoft.com/es-es/library/dd460654.aspx?cs-save-lang=1&cs-lang=vb#code-snippet-6)

class SampleClass

{

 public int sampleMethod(string sampleParam)

 {

 // Insert code here

 }

}

Una clase puede tener varias implementaciones, o *sobrecargas*, del mismo método que se diferencian en el número de parámetros o tipos de parámetros.

Para sobrecargar un método:

C#

[**VB**](http://msdn.microsoft.com/es-es/library/dd460654.aspx?cs-save-lang=1&cs-lang=vb#code-snippet-7)

public int sampleMethod(string sampleParam) {};

public int sampleMethod(int sampleParam) {}

En la mayoría de los casos se declara un método dentro de una definición de clase. Sin embargo, Visual Basic y C# también son compatibles con *métodos de extensión* que le permiten agregar métodos a una clase existente fuera de la definición de la clase.

Para obtener más información, consulte:

* Visual Basic
	+ [Function (Instrucción, Visual Basic)](http://msdn.microsoft.com/es-es/library/sect4ck6.aspx)
	+ [Sub (Instrucción, Visual Basic)](http://msdn.microsoft.com/es-es/library/dz1z94ha.aspx)
	+ [Overloads (Visual Basic)](http://msdn.microsoft.com/es-es/library/ybxdk47f.aspx)
	+ [Métodos de extensión (Visual Basic)](http://msdn.microsoft.com/es-es/library/bb384936.aspx)
* C#
	+ [Métodos (Guía de programación de C#)](http://msdn.microsoft.com/es-es/library/ms173114.aspx)
	+ [Métodos de extensión (Guía de programación de C#)](http://msdn.microsoft.com/es-es/library/bb383977.aspx)

Constructores

Los constructores son métodos de la clase que se ejecutan automáticamente cuando se crea un objeto de un tipo determinado. Normalmente, los constructores inicializan a los miembros de datos del nuevo objeto.Un constructor puede ejecutar sólo una vez cuando se crea una clase. Además, el código en el constructor siempre se ejecuta antes que cualquier otro código en una clase. Sin embargo, puede crear varias sobrecargas del constructor de la misma manera que cualquier otro método.

Para definir un constructor de una clase:

C#

[**VB**](http://msdn.microsoft.com/es-es/library/dd460654.aspx?cs-save-lang=1&cs-lang=vb#code-snippet-8)

public class SampleClass

{

 public SampleClass()

 {

 // Add code here

 }

}

Para obtener más información, consulte:

* Visual Basic
	+ Sub New
	+ [Duración de los objetos: cómo se crean y destruyen (Visual Basic)](http://msdn.microsoft.com/es-es/library/hks5e2k6.aspx)
* C#
	+ [Constructores (Guía de programación de C#)](http://msdn.microsoft.com/es-es/library/ace5hbzh.aspx)

Destructores

Los destructores se utilizan para destruir instancias de clases. En el.NET Framework, el recolector de elementos no utilizados administra automáticamente la asignación y liberación de memoria para los objetos administrados en la aplicación. Sin embargo, puede que aún tenga destructores para limpiar recursos no administrados que crea la aplicación. Puede haber sólo un destructor para una clase.

Para obtener más información acerca de los destructores y recolección de elementos en el.NET Framework, vea [recolección de elementos no utilizados](http://msdn.microsoft.com/es-es/library/0xy59wtx.aspx).

Eventos

Eventos habilitan una clase u objeto para notificarlo a otras clases u objetos algo interesante se produce. La clase que envía (o aumenta) el evento se denomina el *Publisher* y las clases que reciben el evento (o identificador) se denominan *suscriptores*. Para obtener más información acerca de los eventos, cómo se provocan y controlan, vea [Controlar y provocar eventos](http://msdn.microsoft.com/es-es/library/edzehd2t.aspx).

* Visual Basic
	+ Para declarar eventos, utilice el [Event (Instrucción)](http://msdn.microsoft.com/es-es/library/6hwhs172.aspx).
	+ Para provocar eventos, utilice el [RaiseEvent (Instrucción)](http://msdn.microsoft.com/es-es/library/fwd3bwed.aspx).
	+ Para especificar los controladores de eventos de forma declarativa, utilice el [WithEvents (Visual Basic)](http://msdn.microsoft.com/es-es/library/aty3352y.aspx) instrucción y el [Handles (Cláusula, Visual Basic)](http://msdn.microsoft.com/es-es/library/6k46st1y.aspx) cláusula.
	+ Para poder agregar, quitar y cambiar el controlador de eventos asociado a un evento de forma dinámica, utilice el [AddHandler (Instrucción)](http://msdn.microsoft.com/es-es/library/7taxzxka.aspx) y [RemoveHandler (Instrucción)](http://msdn.microsoft.com/es-es/library/3xz97kac.aspx) junto con el[AddressOf (Operador) (Visual Basic)](http://msdn.microsoft.com/es-es/library/y72ewk2b.aspx).
* C#
	+ Para declarar un evento en una clase, utilice el [event (Referencia de C#)](http://msdn.microsoft.com/es-es/library/8627sbea.aspx) palabra clave.
	+ Para provocar un evento, invoque al delegado de eventos.
	+ Para suscribirse a un evento, utilice el **+=** operador; Para cancelar la suscripción de un evento, utilice el **-=** operador.

Clases anidadas

Se llama a una clase definida dentro de otra clase *anidados*. De forma predeterminada, la clase anidada es privada.

C#

[**VB**](http://msdn.microsoft.com/es-es/library/dd460654.aspx?cs-save-lang=1&cs-lang=vb#code-snippet-9)

class Container

{

 class Nested

 {

 // Add code here.

 }

}

Para crear una instancia de la clase anidada, utilice el nombre de la clase contenedora seguido del punto y, a continuación, seguido del nombre de la clase anidada:

C#

[**VB**](http://msdn.microsoft.com/es-es/library/dd460654.aspx?cs-save-lang=1&cs-lang=vb#code-snippet-10)

Container.Nested nestedInstance = new Container.Nested()

Modificadores de acceso y los niveles de acceso

Todas las clases y miembros de clase pueden especificar qué nivel de acceso que se proporcionan a otras clases mediante el uso de *modificadores de acceso*.

Los modificadores de acceso siguientes están disponibles:

|  |  |  |
| --- | --- | --- |
| **Modificador de Visual Basic** | **Modificador C#** | **Definición de** |
| [Public (Visual Basic)](http://msdn.microsoft.com/es-es/library/9dc6we3z.aspx) | [pública](http://msdn.microsoft.com/es-es/library/yzh058ae.aspx) | El tipo o miembro puede obtenerse por cualquier otro código en el mismo ensamblado o en otro ensamblado que hace referencia a ella. |
| [Private (Visual Basic)](http://msdn.microsoft.com/es-es/library/wx059ey1.aspx) | [privada](http://msdn.microsoft.com/es-es/library/st6sy9xe.aspx) | El tipo o miembro sólo puede obtenerse mediante código en la misma clase. |
| [Protected (Visual Basic)](http://msdn.microsoft.com/es-es/library/8050kawf.aspx) | [protegido](http://msdn.microsoft.com/es-es/library/bcd5672a.aspx) | El tipo o miembro sólo puede obtenerse por código de la misma clase o de una clase derivada. |
| [Friend (Visual Basic)](http://msdn.microsoft.com/es-es/library/08w05ey2.aspx) | [interno](http://msdn.microsoft.com/es-es/library/7c5ka91b.aspx) | Puede obtenerse el tipo o miembro cualquier código en el mismo ensamblado, pero no desde otro ensamblado. |
| **Protected Friend** | **protected internal** | El tipo o miembro puede obtenerse por cualquier código en el mismo ensamblado, o por cualquier clase derivada de otro ensamblado. |

Para obtener más información, vea [Niveles de acceso en Visual Basic](http://msdn.microsoft.com/es-es/library/76453kax.aspx) y [Modificadores de acceso (Guía de programación de C#)](http://msdn.microsoft.com/es-es/library/ms173121.aspx).

Instancias de clases

Para crear un objeto, debe crear una instancia de una clase, o crear una instancia de clase.

C#

[**VB**](http://msdn.microsoft.com/es-es/library/dd460654.aspx?cs-save-lang=1&cs-lang=vb#code-snippet-11)

SampleClass sampleObject = new SampleClass();

Después de crear una instancia de una clase, puede asignar valores a propiedades y campos de la instancia y llamar a métodos de clase.

C#

[**VB**](http://msdn.microsoft.com/es-es/library/dd460654.aspx?cs-save-lang=1&cs-lang=vb#code-snippet-12)

// Set a property value.

sampleObject.sampleProperty = "Sample String";

// Call a method.

sampleObject.sampleMethod();

Para asignar valores a propiedades durante el proceso de creación de instancias de clase, utilice a los inicializadores de objeto:

C#

[**VB**](http://msdn.microsoft.com/es-es/library/dd460654.aspx?cs-save-lang=1&cs-lang=vb#code-snippet-13)

// Set a property value.

SampleClass sampleObject = new SampleClass

 { FirstProperty = "A", SecondProperty = "B" };

Para obtener más información, consulte:

* Visual Basic
	+ [New (Operador, Visual Basic)](http://msdn.microsoft.com/es-es/library/77s47661.aspx)
	+ [Inicializadores de objeto: Tipos con nombre y anónimos (Visual Basic)](http://msdn.microsoft.com/es-es/library/bb385125.aspx)
* C#
	+ [new (Operador, Referencia de C#)](http://msdn.microsoft.com/es-es/library/fa0ab757.aspx)
	+ [Inicializadores de objeto y de colección (Guía de programación de C#)](http://msdn.microsoft.com/es-es/library/bb384062.aspx)

Estático (compartido) clases y miembros

Un miembro static (shared en Visual Basic) de la clase es una propiedad, procedimiento o campo que comparten todas las instancias de una clase.

Para definir a un miembro estático (compartido):

C#

[**VB**](http://msdn.microsoft.com/es-es/library/dd460654.aspx?cs-save-lang=1&cs-lang=vb#code-snippet-14)

static class SampleClass

{

 public static string SampleString = "Sample String";

}

Para tener acceso a miembro estático (compartido), utilice el nombre de la clase sin crear un objeto de esta clase:

C#

[**VB**](http://msdn.microsoft.com/es-es/library/dd460654.aspx?cs-save-lang=1&cs-lang=vb#code-snippet-15)

Console.WriteLine(SampleClass.SampleString);

Clases estáticas de (compartidas) en C# y los módulos de Visual Basic tienen estáticos (compartidos) sólo los miembros y no se puede crear una instancia. No los miembros estáticos de (compartidos) también pueden tener acceso no estático (no compartidos) propiedades, campos o métodos

Para obtener más información, consulte:

* Visual Basic
	+ [Shared (Visual Basic)](http://msdn.microsoft.com/es-es/library/zc2b427x.aspx)
	+ [Module (Instrucción)](http://msdn.microsoft.com/es-es/library/aaxss7da.aspx)
* C#
	+ [static (Referencia de C#)](http://msdn.microsoft.com/es-es/library/98f28cdx.aspx)

Tipos anónimos

Los tipos anónimos permiten crear objetos sin tener que escribir una definición de clase para el tipo de datos. En su lugar, el compilador genera una clase para usted. La clase no tiene ningún nombre utilizable y contiene las propiedades especificadas al declarar el objeto.

Para crear una instancia de un tipo anónimo:

C#

[**VB**](http://msdn.microsoft.com/es-es/library/dd460654.aspx?cs-save-lang=1&cs-lang=vb#code-snippet-16)

// sampleObject is an instance of a simple anonymous type.

var sampleObject =

 new { FirstProperty = "A", SecondProperty = "B" };

Para obtener más información, consulte:

* Visual Basic: [Tipos anónimos (Visual Basic)](http://msdn.microsoft.com/es-es/library/bb384767.aspx)
* C#: [Tipos anónimos (Guía de programación de C#)](http://msdn.microsoft.com/es-es/library/bb397696.aspx)

Herencia

La herencia permite crear una nueva clase que se vuelve a utilizar, se extiende y modifica el comportamiento que se define en otra clase. La clase cuyos miembros se heredan se denomina el *clase base*, y la clase que hereda esos miembros se denomina el *clase derivada*. Sin embargo, todas las clases de C# y de Visual Basic heredan implícitamente de la [Object] clase que admite la jerarquía de clases de .NET y proporciona servicios de bajo nivel a todas las clases.

|  |
| --- |
| **Nota Nota** |
| Los lenguajes administrados en el.NET Framework no admiten la herencia múltiple,.e. puede especificar sólo una clase base para una clase derivada. |

Al heredar de una clase base:

C#

[**VB**](http://msdn.microsoft.com/es-es/library/dd460654.aspx?cs-save-lang=1&cs-lang=vb#code-snippet-17)

class DerivedClass:BaseClass{}

De forma predeterminada todas las clases pueden heredar. Sin embargo, puede especificar si una clase no debe utilizarse como una clase base, o crea una clase que se puede utilizar como clase base.

Para especificar que una clase no puede utilizarse como una clase base:

C#

[**VB**](http://msdn.microsoft.com/es-es/library/dd460654.aspx?cs-save-lang=1&cs-lang=vb#code-snippet-18)

public sealed class A { }

Para especificar que una clase puede utilizarse como clase base y no se puede crear una instancia:

C#

[**VB**](http://msdn.microsoft.com/es-es/library/dd460654.aspx?cs-save-lang=1&cs-lang=vb#code-snippet-19)

public abstract class B { }

Para obtener más información, consulte:

* Visual Basic
	+ [Inherits (Instrucción)](http://msdn.microsoft.com/es-es/library/1yk8s7sk.aspx)
	+ [NotInheritable (Visual Basic)](http://msdn.microsoft.com/es-es/library/h278d2d4.aspx)
	+ [MustInherit (Visual Basic)](http://msdn.microsoft.com/es-es/library/aee8f02w.aspx)
* C#
	+ [sealed (Referencia de C#)](http://msdn.microsoft.com/es-es/library/88c54tsw.aspx)
	+ [abstract (Referencia de C#)](http://msdn.microsoft.com/es-es/library/sf985hc5.aspx)

Miembros de reemplazo

De forma predeterminada, una clase derivada hereda a todos los miembros de su clase base. Si desea cambiar el comportamiento del miembro heredado, debe reemplazarlo. Es decir, puede definir una nueva implementación del método, propiedad o evento en la clase derivada.

Los modificadores siguientes se utilizan para controlar cómo se reemplazan las propiedades y métodos:

|  |  |  |
| --- | --- | --- |
| **Modificador de Visual Basic** | **Modificador C#** | **Definición de** |
| [Overridable (Visual Basic)](http://msdn.microsoft.com/es-es/library/zcfd2sa9.aspx) | [virtual (Referencia de C#)](http://msdn.microsoft.com/es-es/library/9fkccyh4.aspx) | Permite a un miembro de clase que sea reemplazada en una clase derivada. |
| [Overrides (Visual Basic)](http://msdn.microsoft.com/es-es/library/k777kzw1.aspx) | [override (Referencia de C#)](http://msdn.microsoft.com/es-es/library/ebca9ah3.aspx) | Reemplaza a un miembro (reemplazable) virtual definido en la clase base. |
| [NotOverridable (Visual Basic)](http://msdn.microsoft.com/es-es/library/ta771xh0.aspx) | No se admite | Impide que un miembro se invalide en una clase heredada. |
| [MustOverride (Visual Basic)](http://msdn.microsoft.com/es-es/library/hyb29zk8.aspx) | [abstract (Referencia de C#)](http://msdn.microsoft.com/es-es/library/sf985hc5.aspx) | Requiere que un miembro de clase que sea reemplazada en la clase derivada. |
| [Shadows (Visual Basic)](http://msdn.microsoft.com/es-es/library/1h3wytf6.aspx) | [new (Modificador, Referencia de C#)](http://msdn.microsoft.com/es-es/library/435f1dw2.aspx) | Oculta un miembro heredado de una clase base |

Interfaces

Las interfaces, como clases, definen un conjunto de propiedades, métodos y eventos. Pero a diferencia de las clases, interfaces no proporcionan implementación. Se implementan las clases y definen como entidades separadas de las clases. Una interfaz representa un contrato, en que una clase que implementa una interfaz debe implementar todos los aspectos de dicha interfaz exactamente como se define.

Para definir una interfaz:

C#

[**VB**](http://msdn.microsoft.com/es-es/library/dd460654.aspx?cs-save-lang=1&cs-lang=vb#code-snippet-20)

interface ISampleInterface

{

 void doSomething();

}

Para implementar una interfaz en una clase:

C#

[**VB**](http://msdn.microsoft.com/es-es/library/dd460654.aspx?cs-save-lang=1&cs-lang=vb#code-snippet-21)

class SampleClass : ISampleInterface

{

 void ISampleInterface.SampleMethod()

 {

 // Method implementation.

 }

}

Para obtener más información, consulte:

* Visual Basic
	+ [Interfaces (Visual Basic)](http://msdn.microsoft.com/es-es/library/28e2e18x.aspx)
	+ [Instrucción Interface (Visual Basic)](http://msdn.microsoft.com/es-es/library/h9xt0sdd.aspx)
	+ [Implements (Instrucción)](http://msdn.microsoft.com/es-es/library/7z6hzchx.aspx)
* C#
	+ [Interfaces (Guía de programación de C#)](http://msdn.microsoft.com/es-es/library/ms173156.aspx)
	+ [interface (Referencia de C#)](http://msdn.microsoft.com/es-es/library/87d83y5b.aspx)

Componentes genéricos

Pueden que las clases, estructuras, interfaces y métodos en el.NET Framework *parámetros de tipo* que definen tipos de objetos que se pueden almacenar o utilizar. El ejemplo más común de genéricos es una colección, donde puede especificar el tipo de objetos que se almacenan en una colección.

Para definir una clase genérica:

C#

[**VB**](http://msdn.microsoft.com/es-es/library/dd460654.aspx?cs-save-lang=1&cs-lang=vb#code-snippet-22)

Public class SampleGeneric<T>

{

 public T Field;

}

Para crear una instancia de una clase genérica:

C#

[**VB**](http://msdn.microsoft.com/es-es/library/dd460654.aspx?cs-save-lang=1&cs-lang=vb#code-snippet-23)

SampleGeneric<string> sampleObject = new SampleGeneric<string>();

sampleObject.Field = "Sample string";

Para obtener más información, consulte:

* [Genéricos en .NET Framework](http://msdn.microsoft.com/es-es/library/ms172192.aspx)
* [Tipos genéricos en Visual Basic (Visual Basic)](http://msdn.microsoft.com/es-es/library/w256ka79.aspx)
* [Genéricos (Guía de programación de C#)](http://msdn.microsoft.com/es-es/library/512aeb7t.aspx)

Delegados

A *delegado* es un tipo que define una firma de método y puede proporcionar una referencia a cualquier método con una firma compatible. Puede invocar (o llamar) al método a través del delegado. Los delegados se utilizan para pasar métodos como argumentos a otros métodos.

|  |
| --- |
| **Nota Nota** |
| Los controladores de eventos no son más que los métodos que se invocan a través de los delegados.Para obtener más información acerca de cómo utilizar delegados en el control de eventos, vea [Eventos y delegados](http://msdn.microsoft.com/es-es/library/17sde2xt.aspx). |

Para crear a un delegado:

C#

[**VB**](http://msdn.microsoft.com/es-es/library/dd460654.aspx?cs-save-lang=1&cs-lang=vb#code-snippet-24)

public delegate void SampleDelegate(string str);

Para crear una referencia a un método que coincida con la firma especificada por el delegado:

C#

[**VB**](http://msdn.microsoft.com/es-es/library/dd460654.aspx?cs-save-lang=1&cs-lang=vb#code-snippet-25)

class SampleClass

{

 // Method that matches the SampleDelegate signature.

 public static void sampleMethod(string message)

 {

 // Add code here.

 }

 // Method that instantiates the delegate.

 void SampleDelegate()

 {

 SampleDelegate sd = sampleMethod;

 sd("Sample string");

 }

}

Para obtener más información, consulte:

* Visual Basic
	+ [Delegados (Visual Basic)](http://msdn.microsoft.com/es-es/library/ms172879.aspx)
	+ [Delegate (Instrucción)](http://msdn.microsoft.com/es-es/library/twsk0311.aspx)
	+ [AddressOf (Operador) (Visual Basic)](http://msdn.microsoft.com/es-es/library/y72ewk2b.aspx)
* C#
	+ [Delegados (Guía de programación de C#)](http://msdn.microsoft.com/es-es/library/ms173171.aspx)
	+ [delegado (Referencia de C#)](http://msdn.microsoft.com/es-es/library/900fyy8e.aspx)