Delegados (Guía de programación de C#)

**Visual Studio 2013**

[Otras versiones](javascript:;)

http://i3.msdn.microsoft.com/Areas/Epx/Content/Images/ImageSprite.png

Personas que lo han encontrado útil: 1 de 1 - [Valorar este tema](http://msdn.microsoft.com/es-es/library/ms173171.aspx#feedback)

Un [delegado](http://msdn.microsoft.com/es-es/library/900fyy8e.aspx) es un tipo que define una firma de método. Al crear instancias de un delegado, puede asociar su instancia con cualquier método mediante una firma compatible. Puede invocar (o llamar) al método a través de la instancia de delegado.

Los delegados se utilizan para pasar métodos como argumentos a otros métodos. Los controladores de eventos no son más que métodos que se invocan a través de delegados. Cree un método personalizado y una clase como un control de Windows podrá llamar al método cuando se produzca un determinado evento. En el siguiente ejemplo se muestra la declaración de un delegado:

C#

public delegate int PerformCalculation(int x, int y);

Cualquier método de cualquier clase o struct accesible que coincida con la firma del delegado, la cual está compuesta por los parámetros y el tipo de valor devuelto, puede asignarse al delegado. El método puede ser un método estático o un método de instancia. Esto permite el cambio mediante programación de las llamadas a métodos y la incorporación de nuevo código en las clases existentes. Si conoce la firma del delegado, puede asignar su propio método.

|  |
| --- |
| **Nota Nota** |
| En el contexto de la sobrecarga de métodos, la firma de un método no incluye el valor devuelto. Sin embargo, en el contexto de los delegados, la firma sí incluye el valor devuelto. En otras palabras, un método debe tener el mismo valor devuelto que el delegado. |

Esta capacidad para hacer referencia a un método como parámetro hace que los delegados sean idóneos para definir métodos de devolución de llamada. Por ejemplo, una referencia a un método que compara dos objetos se puede pasar como argumento a un algoritmo de ordenación. Dado que el código de comparación está en un procedimiento independiente, el algoritmo de ordenación se puede escribir de manera más general.

[Información general sobre delegados](javascript:void(0))

Los delegados tienen las propiedades siguientes:

* Los delegados son similares a los punteros a función de C++, pero poseen seguridad de tipos.
* Los delegados permiten pasar los métodos como parámetros.
* Los delegados pueden utilizarse para definir métodos de devolución de llamada.
* Los delegados pueden encadenarse; por ejemplo, se puede llamar a varios métodos en un solo evento.
* No es necesario que los métodos coincidan exactamente con la firma de delegado. Para obtener más información, vea [Utilizar varianza en delegados (C# y Visual Basic)](http://msdn.microsoft.com/es-es/library/ms173174.aspx).
* La versión 2.0 de C# introdujo el concepto de [métodos anónimos](http://msdn.microsoft.com/es-es/library/0yw3tz5k.aspx), los cuales permiten pasar bloques de código como parámetros en lugar de utilizar métodos definidos independientemente. C# 3.0 introdujo las expresiones lambda como una manera más concisa de escribir bloques de códigos insertados. Tanto los métodos anónimos como las expresiones lambda (en ciertos contextos) se compilan como tipos delegados. En conjunto, estas características se conocen ahora como funciones anónimas. Para obtener más información sobre las expresiones lambda, vea [Funciones anónimas (Guía de programación de C#)](http://msdn.microsoft.com/es-es/library/bb882516.aspx).

Cuándo se utilizan delegados en lugar de interfaces (Guía de programación de C#)

[Otras versiones](javascript:;)

http://i3.msdn.microsoft.com/Areas/Epx/Content/Images/ImageSprite.png

Este tema aún no ha recibido ninguna valoración - [Valorar este tema](http://msdn.microsoft.com/es-es/library/ms173173.aspx#feedback)

Tanto los delegados como las interfaces permiten a un diseñador de clases separar las declaraciones de tipos y la implementación. Una [interfaz](http://msdn.microsoft.com/es-es/library/87d83y5b.aspx) determinada puede ser heredada e implementada por cualquier [clase](http://msdn.microsoft.com/es-es/library/0b0thckt.aspx) o [struct](http://msdn.microsoft.com/es-es/library/ah19swz4.aspx). Se puede crear un [delegado](http://msdn.microsoft.com/es-es/library/900fyy8e.aspx)para un método en cualquier clase, con tal de que el método se ajuste a la firma de método para el delegado. Cualquier objeto puede utilizar una referencia de interfaz o un delegado sin tener conocimiento alguno sobre la clase que implementa el método de interfaz o delegado. Según estas similitudes, ¿cuándo debería un diseñador de clases utilizar un delegado y cuándo una interfaz?

Utilice un delegado cuando:

* Se utilice un modelo de diseño de eventos.
* Se prefiere a la hora de encapsular un método estático.
* El autor de las llamadas no tiene ninguna necesidad de obtener acceso a otras propiedades, métodos o interfaces en el objeto que implementa el método.
* Se desea conseguir una composición sencilla.
* Una clase puede necesitar más de una implementación del método.

Utilice una interfaz cuando:

* Haya un grupo de métodos relacionados a los que se pueda llamar.
* Una clase sólo necesita una implementación del método.
* La clase que utiliza la interfaz deseará convertir esa interfaz en otra interfaz o tipos de clase.
* El método que se va a implementar está vinculado al tipo o identidad de la clase; por ejemplo, métodos de comparación.

Un buen ejemplo del uso de una interfaz de método único en lugar de un delegado es[IComparable](http://msdn.microsoft.com/es-es/library/system.icomparable.aspx) o la versión genérica, [IComparable<T>](http://msdn.microsoft.com/es-es/library/4d7sx9hd.aspx). **IComparable** declara el método[CompareTo](http://msdn.microsoft.com/es-es/library/system.icomparable.compareto.aspx), que devuelve un entero que especifica una relación menor que, igual que o mayor que entre dos objetos del mismo tipo. **IComparable** se puede utilizar como base de un algoritmo de ordenación. Aunque utilizar un método de comparación delegado como la base de un algoritmo de ordenación sería válido, no es lo ideal. Lo ideal es una interfaz de método único, ya que la capacidad de establecer comparaciones pertenece a la clase, y el algoritmo de comparación no cambia en tiempo de ejecución.

# Delegados con métodos con nombre y delegados con métodos anónimos (Guía de programación de C#)

**Visual Studio 2013**

[Otras versiones](javascript:;)

http://i3.msdn.microsoft.com/Areas/Epx/Content/Images/ImageSprite.png

Este tema aún no ha recibido ninguna valoración - [Valorar este tema](http://msdn.microsoft.com/es-es/library/98dc08ac.aspx#feedback)

Se puede asociar un [delegado](http://msdn.microsoft.com/es-es/library/900fyy8e.aspx) a un método con nombre. Cuando se crean instancias de un delegado mediante un método con nombre, el método se pasa como parámetro; por ejemplo:

C#

// Declare a delegate:

delegate void Del(int x);

// Define a named method:

void DoWork(int k) { /\* ... \*/ }

// Instantiate the delegate using the method as a parameter:

Del d = obj.DoWork;

Esto se denomina utilizar un método con nombre. Los delegados creados con un método con nombre pueden encapsular un método [estático](http://msdn.microsoft.com/es-es/library/98f28cdx.aspx) o un método de instancia.Los métodos con nombre son la única forma de crear instancias de un delegado en versiones anteriores de C#. Sin embargo, en una situación en la que crear un método nuevo constituye una sobrecarga no deseada, C# permite crear instancias de un delegado y especificar inmediatamente un bloque de código que el delegado procesará cuando se le llame. El bloque puede contener una expresión lambda o un método anónimo. Para obtener más información, vea [Funciones anónimas (Guía de programación de C#)](http://msdn.microsoft.com/es-es/library/bb882516.aspx).

[Comentarios](javascript:void(0))

El método, que se pasa como parámetro de delegado, debe tener la misma firma que la declaración de delegado.

La instancia de un delegado puede encapsular un método estático o de instancia.

Aunque el delegado puede utilizar un parámetro [out](http://msdn.microsoft.com/es-es/library/t3c3bfhx.aspx), no se recomienda su uso con delegados de eventos de multidifusión porque no se puede saber a qué delegado se va a llamar.

[Ejemplo 1](javascript:void(0))

El siguiente es un ejemplo sencillo de declaración y uso de un delegado. Observe que tanto el delegado, Del, como el método asociado, MultiplyNumbers, tienen la misma firma

C#

// Declare a delegate

delegate void Del(int i, double j);

class MathClass

{

static void Main()

{

MathClass m = new MathClass();

// Delegate instantiation using "MultiplyNumbers"

Del d = m.MultiplyNumbers;

// Invoke the delegate object.

System.Console.WriteLine("Invoking the delegate using 'MultiplyNumbers':");

for (int i = 1; i <= 5; i++)

{

d(i, 2);

}

// Keep the console window open in debug mode.

System.Console.WriteLine("Press any key to exit.");

System.Console.ReadKey();

}

// Declare the associated method.

void MultiplyNumbers(int m, double n)

{

System.Console.Write(m \* n + " ");

}

}

/\* Output:

Invoking the delegate using 'MultiplyNumbers':

2 4 6 8 10

\*/

[Ejemplo 2](javascript:void(0))

En el siguiente ejemplo, un delegado se asigna a métodos estáticos y de instancia y devuelve información específica de cada uno de ellos.

C#

// Declare a delegate

delegate void Del();

class SampleClass

{

public void InstanceMethod()

{

System.Console.WriteLine("A message from the instance method.");

}

static public void StaticMethod()

{

System.Console.WriteLine("A message from the static method.");

}

}

class TestSampleClass

{

static void Main()

{

SampleClass sc = new SampleClass();

// Map the delegate to the instance method:

Del d = sc.InstanceMethod;

d();

// Map to the static method:

d = SampleClass.StaticMethod;

d();

}

}

/\* Output:

A message from the instance method.

A message from the static method.

\*/

Métodos anónimos (Guía de programación de C#)

**Visual Studio 2013**

[Otras versiones](javascript:;)

http://i3.msdn.microsoft.com/Areas/Epx/Content/Images/ImageSprite.png

Personas que lo han encontrado útil: 1 de 1 - [Valorar este tema](http://msdn.microsoft.com/es-es/library/0yw3tz5k.aspx#feedback)

En versiones de C# anteriores a la versión 2.0, la única manera de declarar un[delegado](http://msdn.microsoft.com/es-es/library/900fyy8e.aspx) era utilizar [métodos con nombre](http://msdn.microsoft.com/es-es/library/98dc08ac.aspx). C# 2.0 introdujo los métodos anónimos, mientras que, en C# 3.0 y versiones posteriores, las expresiones lambda reemplazan a los métodos anónimos como la manera preferente de escribir código insertado. No obstante, la información sobre los métodos anónimos de este tema también se aplica a las expresiones lambda. Hay un caso en el que un método anónimo proporciona una funcionalidad que no se encuentra en las expresiones lambda. Los métodos anónimos permiten omitir la lista de parámetros. Esto significa que los métodos anónimos pueden convertirse en delegados con diversas firmas. Esto no es posible con expresiones lambda. Para obtener más información sobre las expresiones lambda, vea[Expresiones lambda (Guía de programación de C#)](http://msdn.microsoft.com/es-es/library/bb397687.aspx).

La creación de métodos anónimos es básicamente una forma de pasar un bloque de código como parámetro de delegado. A continuación se describen dos ejemplos de esto:

C#

// Create a handler for a click event.

button1.Click += delegate(System.Object o, System.EventArgs e)

{ System.Windows.Forms.MessageBox.Show("Click!"); };

C#

// Create a delegate.

delegate void Del(int x);

// Instantiate the delegate using an anonymous method.

Del d = delegate(int k) { /\* ... \*/ };

Mediante los métodos anónimos, se reduce la sobrecarga de codificación a la hora de crear instancias de delegados, ya que no es necesario crear un método independiente.

Por ejemplo, especificar un bloque de código en vez de un delegado puede ser útil en el caso de que la creación de un método pueda suponer una sobrecarga innecesaria.Un buen ejemplo es cuando se inicia un nuevo subproceso. Esta clase crea un subproceso y también contiene el código que el subproceso ejecuta sin crear un método adicional para el delegado.

C#

void StartThread()

{

System.Threading.Thread t1 = new System.Threading.Thread

(delegate()

{

System.Console.Write("Hello, ");

System.Console.WriteLine("World!");

});

t1.Start();

}

[Comentarios](javascript:void(0))

El ámbito de los parámetros de un método anónimo es el *bloque del método anónimo*.

Es un error utilizar una instrucción de salto, como [goto](http://msdn.microsoft.com/es-es/library/13940fs2.aspx), [break](http://msdn.microsoft.com/es-es/library/adbctzc4.aspx) o [continue](http://msdn.microsoft.com/es-es/library/923ahwt1.aspx), en un bloque de método anónimo si el destino está fuera del bloque. También es un error utilizar una instrucción de salto, como **goto**, **break** o **continue**, fuera de un bloque de método anónimo si el destino está dentro del bloque.

Las variables locales y los parámetros cuyo ámbito contiene una declaración de método anónimo se denominan variables *externas* del método anónimo. Por ejemplo, en el segmento de código siguiente, n es una variable externa:

C#

int n = 0;

Del d = delegate() { System.Console.WriteLine("Copy #:{0}", ++n); };

Una referencia a la variable externa n se dice que es *capturado* cuando se crea el delegado. A diferencia de las variables locales, la duración de una variable capturada se extiende hasta que los delegados que hacen referencia a los métodos anónimos son elegibles para la recolección.

Un método anónimo no puede tener acceso a los parámetros [ref](http://msdn.microsoft.com/es-es/library/14akc2c7.aspx) u [out](http://msdn.microsoft.com/es-es/library/t3c3bfhx.aspx) de un ámbito externo.

No se puede obtener acceso a código no seguro dentro del *bloque de método anónimo*.

No se permite el uso de métodos anónimos en el lado izquierdo del operador [is](http://msdn.microsoft.com/es-es/library/scekt9xw.aspx).

[Ejemplo](javascript:void(0))

El ejemplo siguiente muestra las dos maneras de crear instancias de un delegado:

* Asociar el delegado a un método anónimo.
* Asociar el delegado a un método con nombre (DoWork).

En cada uno de los casos, se muestra un mensaje cuando se invoca al delegado.

C#

// Declare a delegate.

delegate void Printer(string s);

class TestClass

{

static void Main()

{

// Instantiate the delegate type using an anonymous method.

Printer p = delegate(string j)

{

System.Console.WriteLine(j);

};

// Results from the anonymous delegate call.

p("The delegate using the anonymous method is called.");

// The delegate instantiation using a named method "DoWork".

p = new Printer(TestClass.DoWork);

// Results from the old style delegate call.

p("The delegate using the named method is called.");

}

// The method associated with the named delegate.

static void DoWork(string k)

{

System.Console.WriteLine(k);

}

}

/\* Output:

The delegate using the anonymous method is called.

The delegate using the named method is called.

\*/

# Cómo: Combinar delegados (delegados de multidifusión) (Guía de programación de C#)

**Visual Studio 2013**

[Otras versiones](javascript:;)

http://i3.msdn.microsoft.com/Areas/Epx/Content/Images/ImageSprite.png

Personas que lo han encontrado útil: 1 de 1 - [Valorar este tema](http://msdn.microsoft.com/es-es/library/ms173175.aspx#feedback)

En este ejemplo se muestra cómo se crean delegados de multidifusión. Una propiedad útil de los objetos [delegate](http://msdn.microsoft.com/es-es/library/900fyy8e.aspx) es que se pueden asignar varios objetos a una única instancia del delegado con el operador **+**. El delegado de multidifusión contiene una lista de los delegados asignados. Cuando se llama al delegado de multidifusión, este invoca los delegados de la lista, en orden. Solo pueden combinarse delegados del mismo tipo.

El operador **-** se puede usar para quitar un delegado componente de un delegado de multidifusión.

[Ejemplo](javascript:void(0))

C#

using System;

// Define a custom delegate that has a string parameter and returns void.

delegate void CustomDel(string s);

class TestClass

{

// Define two methods that have the same signature as CustomDel.

static void Hello(string s)

{

System.Console.WriteLine(" Hello, {0}!", s);

}

static void Goodbye(string s)

{

System.Console.WriteLine(" Goodbye, {0}!", s);

}

static void Main()

{

// Declare instances of the custom delegate.

CustomDel hiDel, byeDel, multiDel, multiMinusHiDel;

// In this example, you can omit the custom delegate if you

// want to and use Action<string> instead.

//Action<string> hiDel, byeDel, multiDel, multiMinusHiDel;

// Create the delegate object hiDel that references the

// method Hello.

hiDel = Hello;

// Create the delegate object byeDel that references the

// method Goodbye.

byeDel = Goodbye;

// The two delegates, hiDel and byeDel, are combined to

// form multiDel.

multiDel = hiDel + byeDel;

// Remove hiDel from the multicast delegate, leaving byeDel,

// which calls only the method Goodbye.

multiMinusHiDel = multiDel - hiDel;

Console.WriteLine("Invoking delegate hiDel:");

hiDel("A");

Console.WriteLine("Invoking delegate byeDel:");

byeDel("B");

Console.WriteLine("Invoking delegate multiDel:");

multiDel("C");

Console.WriteLine("Invoking delegate multiMinusHiDel:");

multiMinusHiDel("D");

}

}

/\* Output:

Invoking delegate hiDel:

Hello, A!

Invoking delegate byeDel:

Goodbye, B!

Invoking delegate multiDel:

Hello, C!

Goodbye, C!

Invoking delegate multiMinusHiDel:

Goodbye, D!

\*/

# Utilizar varianza en delegados (C# y Visual Basic)

**Visual Studio 2013**

[Otras versiones](javascript:;)

http://i3.msdn.microsoft.com/Areas/Epx/Content/Images/ImageSprite.png

Este tema aún no ha recibido ninguna valoración - [Valorar este tema](http://msdn.microsoft.com/es-es/library/ms173174.aspx#feedback)

Al asignar un método a un delegado, la *covarianza* y la *contravarianza* proporcionan flexibilidad para hacer coincidir un tipo de delegado con una firma de método. La covarianza permite que un método tenga un tipo de valor devuelto más derivado que lo que se define en el delegado. La contravarianza permite un método que tiene tipos de parámetro menos derivados que los del tipo de delegado.

[Ejemplo 1: Covarianza](javascript:void(0))

### ms173174.collapse_all(es-es,VS.120).gifDescripción

Este ejemplo muestra cómo se pueden utilizar los delegados con métodos que tienen tipos de valor devueltos que se derivan del tipo de valor devuelto en la firma de delegado. El tipo de datos devuelto por DogsHandler es de tipo Dogs, que se deriva del tipo Mammals definido en el delegado.

### ms173174.collapse_all(es-es,VS.120).gifCódigo

C#

[**VB**](http://msdn.microsoft.com/es-es/library/ms173174.aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1)

class Mammals{}

class Dogs : Mammals{}

class Program

{

// Define the delegate.

public delegate Mammals HandlerMethod();

public static Mammals MammalsHandler()

{

return null;

}

public static Dogs DogsHandler()

{

return null;

}

static void Test()

{

HandlerMethod handlerMammals = MammalsHandler;

// Covariance enables this assignment.

HandlerMethod handlerDogs = DogsHandler;

}

}

[Ejemplo 2: Contravarianza](javascript:void(0))

### ms173174.collapse_all(es-es,VS.120).gifDescripción

Este ejemplo muestra cómo se pueden utilizar los delegados con métodos que tienen parámetros de un tipo que son tipos base del tipo de parámetro de la firma de delegado. Con la contravarianza, puede utilizar un controlador de eventos en lugar de controladores independientes. Por ejemplo, puede crear un controlador de eventos que acepte un parámetro de entrada EventArgs y lo utilice con un eventoButton.MouseClick que envíe un tipo MouseEventArgs como parámetro, y también con un evento TextBox.KeyDown que envíe un parámetro KeyEventArgs.

### ms173174.collapse_all(es-es,VS.120).gifCódigo

C#

[**VB**](http://msdn.microsoft.com/es-es/library/ms173174.aspx?cs-save-lang=1&cs-lang=vb#code-snippet-2)

// Event hander that accepts a parameter of the EventArgs type.

private void MultiHandler(object sender, System.EventArgs e)

{

label1.Text = System.DateTime.Now.ToString();

}

public Form1()

{

InitializeComponent();

// You can use a method that has an EventArgs parameter,

// although the event expects the KeyEventArgs parameter.

this.button1.KeyDown += this.MultiHandler;

// You can use the same method

// for an event that expects the MouseEventArgs parameter.

this.button1.MouseClick += this.MultiHandler;

}

[Vea también](javascript:void(0))

#### Referencia

[Usar la varianza para los delegados genéricos Func y Action (C# y Visual Basic)](http://msdn.microsoft.com/es-es/library/dd465122.aspx)

#### Conceptos

[Varianza en delegados (C# y Visual Basic)](http://msdn.microsoft.com/es-es/library/dd233060.aspx)

Cómo: Declarar un delegado, crear instancias del mismo y utilizarlo (Guía de programación de C#)

**Visual Studio 2013**

[Otras versiones](javascript:;)

http://i3.msdn.microsoft.com/Areas/Epx/Content/Images/ImageSprite.png

Este tema aún no ha recibido ninguna valoración - [Valorar este tema](http://msdn.microsoft.com/es-es/library/ms173176.aspx#feedback)

En C# 1.0 y versiones posteriores, los delegados pueden declararse tal y como se muestra en el ejemplo siguiente.

C#

// Declare a delegate.

delegate void Del(string str);

// Declare a method with the same signature as the delegate.

static void Notify(string name)

{

Console.WriteLine("Notification received for: {0}", name);

}

C#

// Create an instance of the delegate.

Del del1 = new Del(Notify);

C# 2.0 proporciona una manera más sencilla de escribir la declaración anterior, tal y como se muestra en el ejemplo siguiente.

C#

// C# 2.0 provides a simpler way to declare an instance of Del.

Del del2 = Notify;

En C# 2.0 y versiones posteriores, también es posible usar un método anónimo para declarar e inicializar un [delegado](http://msdn.microsoft.com/es-es/library/900fyy8e.aspx), tal y como se muestra en el ejemplo siguiente.

C#

// Instantiate Del by using an anonymous method.

Del del3 = delegate(string name)

{ Console.WriteLine("Notification received for: {0}", name); };

En C# 3.0 y versiones posteriores, también se pueden crear declaraciones e instancias de los delegados usando una expresión lambda, tal y como se muestra en el ejemplo siguiente.

C#

// Instantiate Del by using a lambda expression.

Del del4 = name => { Console.WriteLine("Notification received for: {0}", name); };

Para obtener más información, vea [Expresiones lambda (Guía de programación de C#)](http://msdn.microsoft.com/es-es/library/bb397687.aspx).

El siguiente ejemplo ilustra la declaración, creación de instancias y uso de un delegado.La clase BookDB encapsula una base de datos de los libros de una librería. Expone un método ProcessPaperbackBooks, el cual busca todos los libros en edición rústica de la base de datos y llama a un delegado para cada uno. El tipo **delegate** que se utiliza se denomina ProcessBookDelegate. La clase Test utiliza esta clase para imprimir los títulos y el precio medio de los libros en edición rústica.

El uso de delegados promueve una buena separación de la funcionalidad entre la base de datos de la librería y el código del programa cliente. El código del cliente no tiene conocimiento de cómo están almacenados los libros ni de cómo busca el código de la librería de los libros en rústica. El código de la librería no conoce qué procesamiento se realiza sobre los libros en rústica después de encontrarlos.

[Ejemplo](javascript:void(0))

C#

// A set of classes for handling a bookstore:

namespace Bookstore

{

using System.Collections;

// Describes a book in the book list:

public struct Book

{

public string Title; // Title of the book.

public string Author; // Author of the book.

public decimal Price; // Price of the book.

public bool Paperback; // Is it paperback?

public Book(string title, string author, decimal price, bool paperBack)

{

Title = title;

Author = author;

Price = price;

Paperback = paperBack;

}

}

// Declare a delegate type for processing a book:

public delegate void ProcessBookDelegate(Book book);

// Maintains a book database.

public class BookDB

{

// List of all books in the database:

ArrayList list = new ArrayList();

// Add a book to the database:

public void AddBook(string title, string author, decimal price, bool paperBack)

{

list.Add(new Book(title, author, price, paperBack));

}

// Call a passed-in delegate on each paperback book to process it:

public void ProcessPaperbackBooks(ProcessBookDelegate processBook)

{

foreach (Book b in list)

{

if (b.Paperback)

// Calling the delegate:

processBook(b);

}

}

}

}

// Using the Bookstore classes:

namespace BookTestClient

{

using Bookstore;

// Class to total and average prices of books:

class PriceTotaller

{

int countBooks = 0;

decimal priceBooks = 0.0m;

internal void AddBookToTotal(Book book)

{

countBooks += 1;

priceBooks += book.Price;

}

internal decimal AveragePrice()

{

return priceBooks / countBooks;

}

}

// Class to test the book database:

class TestBookDB

{

// Print the title of the book.

static void PrintTitle(Book b)

{

System.Console.WriteLine(" {0}", b.Title);

}

// Execution starts here.

static void Main()

{

BookDB bookDB = new BookDB();

// Initialize the database with some books:

AddBooks(bookDB);

// Print all the titles of paperbacks:

System.Console.WriteLine("Paperback Book Titles:");

// Create a new delegate object associated with the static

// method Test.PrintTitle:

bookDB.ProcessPaperbackBooks(PrintTitle);

// Get the average price of a paperback by using

// a PriceTotaller object:

PriceTotaller totaller = new PriceTotaller();

// Create a new delegate object associated with the nonstatic

// method AddBookToTotal on the object totaller:

bookDB.ProcessPaperbackBooks(totaller.AddBookToTotal);

System.Console.WriteLine("Average Paperback Book Price: ${0:#.##}",

totaller.AveragePrice());

}

// Initialize the book database with some test books:

static void AddBooks(BookDB bookDB)

{

bookDB.AddBook("The C Programming Language", "Brian W. Kernighan and Dennis M. Ritchie", 19.95m, true);

bookDB.AddBook("The Unicode Standard 2.0", "The Unicode Consortium", 39.95m, true);

bookDB.AddBook("The MS-DOS Encyclopedia", "Ray Duncan", 129.95m, false);

bookDB.AddBook("Dogbert's Clues for the Clueless", "Scott Adams", 12.00m, true);

}

}

}

/\* Output:

Paperback Book Titles:

The C Programming Language

The Unicode Standard 2.0

Dogbert's Clues for the Clueless

Average Paperback Book Price: $23.97

\*/

[Programación eficaz](javascript:void(0))

* Declarar un delegado.

La instrucción siguiente declara un nuevo tipo delegado.

C#

public delegate void ProcessBookDelegate(Book book);

Cada tipo delegado describe el número y tipo de los argumentos, así como el tipo del valor devuelto de los métodos que puede encapsular. Cuando se necesita un nuevo conjunto de tipos de argumentos o de valor devuelto, se debe declarar un nuevo tipo delegado.

* Crear instancias de un delegado.

Una vez declarado un tipo delegado, debe crearse un objeto delegado y asociarlo con un determinado método. En el ejemplo anterior, esto se hace pasando el método PrintTitle al método ProcessPaperbackBooks como en el ejemplo siguiente:

C#

bookDB.ProcessPaperbackBooks(PrintTitle);

Esto crea un nuevo objeto delegado asociado con el método[estático](http://msdn.microsoft.com/es-es/library/98f28cdx.aspx)Test.PrintTitle. De igual forma, el método no estáticoAddBookToTotal del objeto totaller se pasa como en el ejemplo siguiente:

C#

bookDB.ProcessPaperbackBooks(totaller.AddBookToTotal);

En ambos casos, este nuevo objeto delegado se pasa al métodoProcessPaperbackBooks.

Una vez que se crea el delegado, el método con el que está asociado no cambia nunca; los objetos delegados son inmutables.

* Llamar a un delegado.

Una vez creado un objeto delegado, éste se pasa normalmente a otro código que llamará al delegado. La llamada a un objeto delegado se realiza mediante el nombre del objeto delegado, seguido por los argumentos entre paréntesis que se pasarán al delegado. A continuación, se muestra un ejemplo de una llamada a delegado:

C#

processBook(b);

Se puede llamar a un delegado de forma sincrónica, como en este ejemplo, o de forma asincrónica, utilizando los métodos **BeginInvoke** y **EndInvoke**.

abstract (Referencia de C#)

**Visual Studio 2013**

[Otras versiones](javascript:;)

http://i3.msdn.microsoft.com/Areas/Epx/Content/Images/ImageSprite.png

Este tema aún no ha recibido ninguna valoración - [Valorar este tema](http://msdn.microsoft.com/es-es/library/sf985hc5.aspx#feedback)

El modificador **abstract** indica que el elemento objeto de la modificación tiene una implementación incompleta o que le falta una implementación. El modificador abstract se puede usar con clases, métodos, propiedades, indizadores y eventos. Use el modificador **abstract** en una declaración de clase para indicar que la clase sólo se puede utilizar como clase base de otras clases. Los miembros que están marcados como abstractos o que se incluyen en una clase abstracta, deben ser implementados por clases derivadas de la clase abstracta.

[Ejemplo](javascript:void(0))

En este ejemplo, la clase Square debe proporcionar una implementación de Areaporque deriva de ShapesClass:

C#

abstract class ShapesClass

{

abstract public int Area();

}

class Square : ShapesClass

{

int side = 0;

public Square(int n)

{

side = n;

}

// Area method is required to avoid

// a compile-time error.

public override int Area()

{

return side \* side;

}

static void Main()

{

Square sq = new Square(12);

Console.WriteLine("Area of the square = {0}", sq.Area());

}

interface I

{

void M();

}

abstract class C : I

{

public abstract void M();

}

}

// Output: Area of the square = 144

Las clases de tipo abstract presentan las siguientes características:

* No se pueden crear instancias de una clase abstracta.
* Una clase abstracta puede contener descriptores de acceso y métodos abstractos.
* No se puede modificar una clase abstracta con el modificador [sealed (Referencia de C#)](http://msdn.microsoft.com/es-es/library/88c54tsw.aspx) porque los dos modificadores tienen significados opuestos. El modificador **sealed** evita la herencia de una clase y el modificador **abstract** requiere la herencia de una clase.
* Una clase no abstracta derivada de una clase abstracta debe incluir implementaciones reales de todos los descriptores de acceso y métodos abstractos heredados.

Utilice el modificador **abstract** en una declaración de método o propiedad para indicar que el método o la propiedad no contienen implementación.

Los métodos abstractos presentan las siguientes características:

* Un método abstracto es, implícitamente, un método virtual.
* Las declaraciones de métodos abstractos sólo se permiten en clases abstractas.
* Debido a que una declaración de método abstracto no proporciona una implementación, no existe cuerpo del método; la declaración de método finaliza simplemente con un punto y coma y sin llaves ({ }) después de la firma.Por ejemplo:
* public abstract void MyMethod();

La implementación la proporciona un método de reemplazo[override (Referencia de C#)](http://msdn.microsoft.com/es-es/library/ebca9ah3.aspx), que es miembro de una clase no abstracta.

* Utilizar los modificadores [static](http://msdn.microsoft.com/es-es/library/98f28cdx.aspx) o [virtual](http://msdn.microsoft.com/es-es/library/9fkccyh4.aspx) en una declaración de método abstracto produce un error.

Las propiedades abstractas funcionan como los métodos abstractos, salvo las diferencias en la sintaxis de las declaraciones y llamadas.

* Es incorrecto utilizar el modificador **abstract** para una propiedad estática.
* Una propiedad abstracta heredada se puede reemplazar en una clase derivada si se incluye una declaración de propiedad que utilice el modificador[override](http://msdn.microsoft.com/es-es/library/ebca9ah3.aspx).

Para obtener más información sobre las clases abstractas, vea [Clases y miembros de clase abstractos y sellados (Guía de programación de C#)](http://msdn.microsoft.com/es-es/library/ms173150.aspx).

Una clase abstracta debe proporcionar implementaciones para todos los miembros de la interfaz.

Una clase abstracta que implementa una interfaz podría asignar los métodos de la interfaz a métodos abstractos. Por ejemplo:

C#

interface I

{

void M();

}

abstract class C : I

{

public abstract void M();

}

En este ejemplo, la clase DerivedClass se deriva de una clase abstracta BaseClass.La clase abstracta contiene un método abstracto, AbstractMethod, y dos propiedades abstractas, X y Y.

C#

abstract class BaseClass // Abstract class

{

protected int \_x = 100;

protected int \_y = 150;

public abstract void AbstractMethod(); // Abstract method

public abstract int X { get; }

public abstract int Y { get; }

}

class DerivedClass : BaseClass

{

public override void AbstractMethod()

{

\_x++;

\_y++;

}

public override int X // overriding property

{

get

{

return \_x + 10;

}

}

public override int Y // overriding property

{

get

{

return \_y + 10;

}

}

static void Main()

{

DerivedClass o = new DerivedClass();

o.AbstractMethod();

Console.WriteLine("x = {0}, y = {1}", o.X, o.Y);

}

}

// Output: x = 111, y = 161

En el ejemplo anterior, si intenta crear una instancia de la clase abstracta mediante una instrucción como la siguiente:

BaseClass bc = new BaseClass(); // Error

se obtendrá un error que indica que el compilador no puede crear una instancia de la clase abstracta 'BaseClass'.