Introducción a C#

Contenido

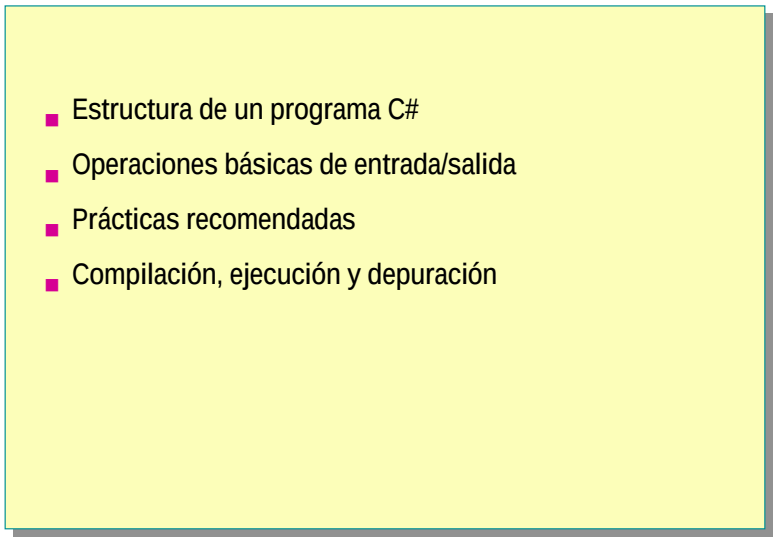
Notas para el instructor

- Este módulo presenta a los estudiantes el lenguaje de programación C#. Los estudiantes aprenderán los elementos básicos de un programa C# elemental. También se discutirán buenas prácticas y estilo de programación. Finalmente, los estudiantes aprenderán a usar Microsoft® Visual Studio® para editar, compilar, ejecutar y depurar un programa C#.
- Al final de este módulo, los estudiantes serán capaces de:
- Explicar la estructura de un programa C# sencillo.
- Utilizar la clase **Console** del espacio de nombres **System** para realizar operaciones básicas de entrada/salida.
- Tratar excepciones en un programa C#.
- Generar documentación en lenguaje de marcado extensible (XML) para un programa C#.
- Compilar, vincular y ejecutar un programa C#.
- Utilizar el Visual Studio Debugger para seguir paso a paso un programa C#.

◆Notas generales

Objetivo del tema
Ofrecer una introducción a los contenidos y objetivos del módulo.

Explicación previa
En este módulo aprenderá los elementos básicos de C# y cómo compilar, vincular y ejecutar un programa C#.



En este módulo estudiará la estructura básica de un programa C# analizando un ejemplo de trabajo sencillo. Aprenderá a usar la clase **Console** para realizar algunas operaciones básicas de entrada y salida. Aprenderá también algunas prácticas recomendadas para el tratamiento de errores y la documentación del código, y finalmente compilará, ejecutará y compilará un programa C#.

Al final de este módulo, usted será capaz de:

- Explicar la estructura de un programa C# sencillo.
- Utilizar la clase **Console** del espacio de nombres **System** para realizar operaciones básicas de entrada/salida.
- Tratar excepciones en un programa C#.
- Generar documentación en lenguaje de marcado extensible (XML) para un programa C#.
- Compilar y ejecutar un programa C#.
- Utilizar el depurador para seguir paso a paso la ejecución de un programa.

Estructura de un programa C#

Objetivo del tema
Ofrecer una introducción a los temas tratados en esta sección.

Explicación previa
En esta sección estudiará la estructura básica de un programa C#.

- Hola, mundo
- La clase
- El método Main
- La sentencia using y el espacio de nombres System
- Demostración: Uso de Visual Studio para crear un programa C#

Recomendación al profesor
En esta lección se hacen muchas comparaciones entre C# y otros lenguajes con los que los estudiantes pueden estar familiarizados. Es recomendable señalar las similitudes y las pequeñas (y no tan pequeñas) diferencias.

En esta lección estudiará la estructura básica de un programa C# analizando un programa simple que contiene todas las características esenciales. También aprenderá a usar Microsoft® Visual Studio® para crear y editar un programa C#.

Hola, mundo

Objetivo del tema
Mostrar un programa C# que funciona.

Explicación previa
El primer programa que se suele escribir en un lenguaje nuevo es el inevitable Hola, mundo.

```
using System;

class Hola
{
  public static void Main()
  {
    Console.WriteLine("Hola, mundo”);
  }
}
```

Recomendación al profesor
Éste es el estilo, diseño y definición de Main que emplea Visual Studio para crear un nuevo programa C#.

Puede escribir el código, compilarlo y ejecutarlo desde la línea de comandos.

El primer programa que se suele escribir cuando se aprende un lenguaje nuevo es el inevitable Hola, mundo. En este módulo tendrá la oportunidad de examinar la versión en C# de este primer programa.

El código de ejemplo en la transparencia contiene todos los elementos esenciales de un programa C# y es muy fácil de probar. Lo único que hace cuando se ejecuta desde la línea de comandos es mostrar el siguiente mensaje:

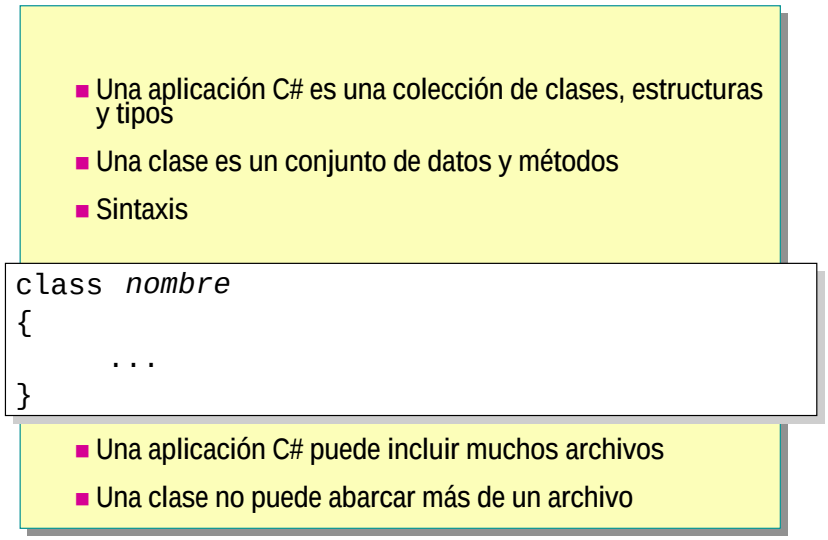
```
Hola, mundo
```

En los temas que siguen analizaremos este programa simple para aprender más sobre los componentes de un programa C#.

La clase

Objetivo del tema
Destacar que todas las aplicaciones C# son una colección de clases.

Explicación previa
Una aplicación C# es una colección de una o más clases.



Recomendación al profesor
No inicie todavía una discusión completa sobre la definición de una clase. Limítese a la explicación sencilla que se da en las notas de los estudiantes.

En C#, una aplicación es una colección de una o más clases, estructuras de datos y otros tipos. En este módulo se define una clase como un conjunto de datos combinados con métodos (o funciones) que pueden manipular esos datos. En módulos posteriores aprenderemos más sobre clases y todo lo que pueden ofrecer al programador de C#.

Un examen del código para la aplicación Hola, mundo revela que hay una sola clase llamada **Hola**. Esta clase se introduce con la palabra clave **class**. Después del nombre de la clase hay una llave de apertura ({). Todo lo que hay hasta la correspondiente llave de cierre (}) forma parte de la clase.

Las clases para una aplicación C# se pueden extender a uno o más archivos. Es posible poner varias clases en un archivo, pero una sola clase no puede abarcar más de un archivo.

Nota para programadores en Java No es necesario que el nombre del archivo de la aplicación coincida con el nombre de la clase.

Nota para programadores en C++ C# no distingue entre la definición y el uso de una clase de la misma forma que C++. No existe el concepto de archivo de definición (.hpp). Todo el código para la clase se escribe en un solo archivo.

El método Main

Objetivo del tema
Explicar dónde comienza una aplicación C# cuando se ejecuta.

Explicación previa
Cuando se ejecuta una aplicación C#, la ejecución se inicia con el método Main.

- Al escribir Main hay que:
  - Utilizar una "M" mayúscula, como en "Main"
  - Designar un Main como el punto de entrada al programa
  - Declarar Main como public static void Main
- Un Main puede pertenecer a múltiple clases
- La aplicación termina cuando Main acaba o ejecuta un return

Consejo para el profesor
Al tratar este tema, haga hincapié en que muchos de los puntos discutidos se tratarán en detalle más adelante. El objetivo de este tema es simplemente destacar las reglas básicas del método Main.

Recomendación al profesor
Es posible poner Main en una estructura en lugar de una clase. Las estructuras se tratarán más adelante en este módulo.

Todas las aplicaciones tienen que empezar por algún sitio. Cuando se ejecuta una aplicación C#, la ejecución se inicia en el método llamado **Main**. Si está acostumbrado a programar en C, C++ o incluso en Java, este concepto le resultará familiar.

El lenguaje C# distingue entre mayúsculas y minúsculas. **Main** debe estar escrito con una "M" mayúscula y con las demás letras en minúsculas.

Aunque en una aplicación C# puede haber muchas clases, no puede haber más que un punto de entrada. Es posible tener muchas clases con **Main** en la misma aplicación, pero se ejecutará sólo un **Main**. Al compilar la aplicación hay que especificar cuál se va a utilizar.

También es importante la firma de **Main**. Si se emplea Visual Studio, se creará automáticamente como **static void** (como veremos más adelante en el curso). No se debe cambiar la firma si no hay una buena razón para hacerlo.

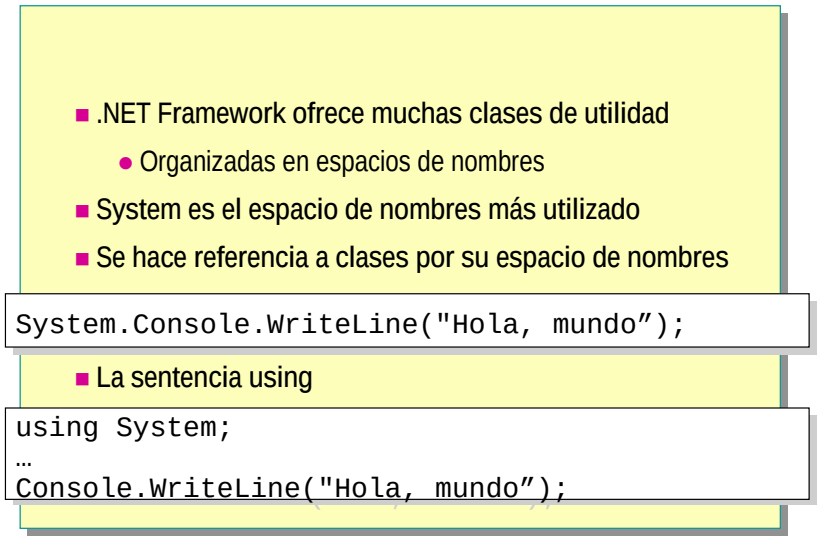
Consejo Hasta cierto punto es posible cambiar la firma, pero debe ser siempre static, ya que de lo contrario es posible que el compilador no la reconozca como punto de entrada a la aplicación.

La aplicación se ejecuta hasta llegar al final de **Main** o hasta que **Main** ejecuta una instrucción **return**.

La sentencia using y el espacio de nombres System

Objetivo del tema
Explicar el empleo de la sentencia **using** para acceder a espacios de nombres.

Explicación previa
C# incluye muchas clases de utilidad que están organizadas en espacios de nombres.



Recomendación al profesor
Los espacios de nombres se tratarán con detalle en el módulo 11, "Agregación, espacios de nombres y ámbito avanzado", del curso 2124C, *Programación en C#*. La información que se da aquí es suficiente para entender la sentencia **using**.

Como parte de Microsoft .NET Framework, C# incluye muchas clases de utilidad que realizan una gran variedad de operaciones útiles. Estas clases están organizadas en *espacios de nombres*, que son conjuntos de clases relacionadas. Un espacio de nombres también puede contener otros espacios de nombres.

.NET Framework está compuesto por muchos espacios de nombres, el más importante de los cuales se llama **System**. El nombre de espacios **System** contiene las clases que emplean la mayor parte de las aplicaciones para interactuar con el sistema operativo. Las clases más utilizadas son las de entrada y salida (E/S). Como ocurre en muchos otros lenguajes, C# no tiene funciones propias de E/S y por tanto depende del sistema operativo para ofrecer una interfaz compatible con C#.

Para hacer referencia a objetos en espacios de nombres se utiliza un prefijo explícito con el identificador del espacio de nombres. Por ejemplo, el espacio de nombres **System** contiene la clase **Console**, que a su vez contiene varios métodos, como **WriteLine**. Se puede acceder al método **WriteLine** de la clase **Console** de la siguiente manera:

```
System.Console.WriteLine("Hola, mundo");
```

Sin embargo, el uso de un nombre completo para referirse a objetos puede resultar poco manejable y propicio a errores. Para hacerlo más sencillo, se puede especificar un espacio de nombres poniendo una sentencia **using** al comienzo de la aplicación, antes de la definición de la primera clase. Una sentencia **using** especifica el espacio de nombres que se examinara si una clase no está definida explícitamente en la aplicación. Es posible poner más de una sentencia **using** en el archivo de origen, pero todas tienen que ir al principio del archivo.

Recomendación al profesor
Asegúrese de que los programadores de C y C++ entienden la diferencia entre la sentencia using y la declaración #include. Los desarrolladores de Java estarán más familiarizados con este punto, ya que es similar a la sentencia import.

Con la sentencia **using** se puede escribir el código anterior de la siguiente manera:

```
using System;
...
Console.WriteLine("Hola, mundo");
```

En la aplicación Hola, mundo, la clase **Console** no está definida explícitamente. Cuando se compila la aplicación, el compilador busca **Console** en el espacio de nombres **System** y genera código que hace referencia al nombre completo **System.Console**.

Nota Las clases del espacio de nombres **System**, así como las demás funciones básicas a las que se accede en tiempo de ejecución, residen en un ensamblado llamado mscorlib.dll que es el que se utiliza por defecto. Es posible referirse a clases de otros ensamblados, pero para ello hay que especificar las ubicaciones y los nombres de esos ensamblados al compilar la aplicación.

Uso de Visual Studio para crear un programa C#

Cómo crear una aplicación C#

1. Inicie Visual Studio .NET.
2. En el menú **File** (Archivo), señale **New** (Nuevo) y pulse **Project** (Proyecto).
3. En el cuadro de diálogo **New Project** (Nuevo proyecto), escriba la información indicada en la siguiente tabla y pulse **OK**.

Elemento	Valor
Tipo de proyecto (vista de árbol)	**Visual C# Projects**
Plantillas (icono)	**Console Application**
Nombre	**Hola**
Ubicación	**C:\temp**

4. Una vez generado el proyecto, señale y discuta las siguientes características de Visual Studio:
   - La ventana Solution Explorer (Explorador de soluciones)
     i. Cierre la ventana de código Class1.cs.
     ii. Cambie el nombre de Class1.cs a Hola.cs.
     iii. Haga doble clic en Hola.cs para volver a abrir la ventana de código.
   - La ventana Properties (Propiedades)
   - Las barras de herramientas
   - El menú **View** (Ver)
   - El menú **Build** (Compilar)
   - El menú **Debug** (Depurar)
   - El menú **Help** (Ayuda)
5. Utilizando la ventana de código Hola.cs, señale y discuta lo siguiente:
   - El espacio de nombres **Hola**

     Menciones que los espacios de nombres se discutirán más adelante. Podría borrar esta línea y las llaves correspondientes, pero por el momento déjelas como están.
   - La sentencia **using**
   - Los comentarios de XML

     Use esta sección para describir brevemente el programa. Los comentarios se discutirán en detalle más adelante.
   - La definición de clase

     El nombre por defecto de la clase es **Class1**. Cámbielo a **Demonstrator**.
   - El método **Main**

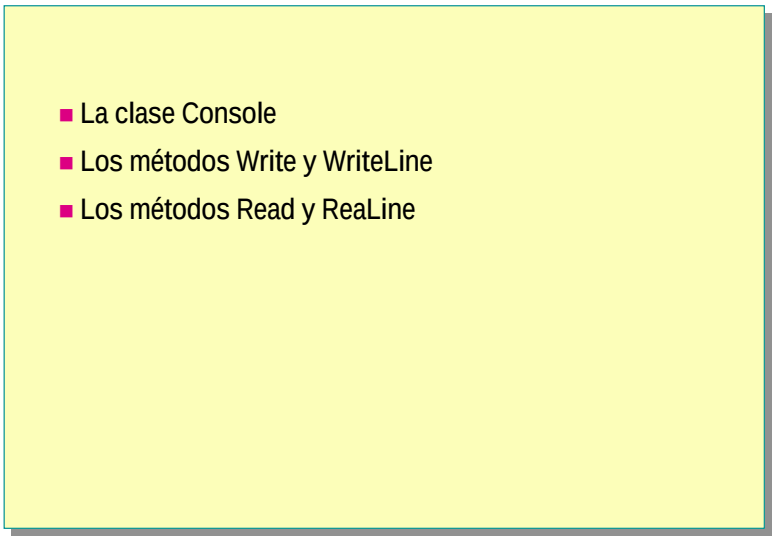
     Escriba el siguiente código:

     ```
     Console.WriteLine("Hola, mundo");
     ```
   - Microsoft IntelliSense®
6. En el menú **File**, seleccione **Save All** (Guardar todo).
7. Cierre Visual Studio.

◆ Operaciones básicas de entrada/salida

Objetivo del tema
Ofrecer una introducción a los temas tratados en esta sección.

Explicación previa
En esta lección aprenderá a realizar en C# operaciones básicas de E/S orientadas a la consola.



En esta lección aprenderá a realizar en C# operaciones de entrada/salida basadas en comandos utilizando la clase **Console**. También aprenderá a mostrar información con los métodos **Write** y **WriteLine**, así como a obtener información introducida desde el teclado con los métodos **Read** y **ReadLine**.

La clase Console

Objetivo del tema
Explicar el empleo de la clase Console.

Explicación previa
Puede emplear métodos de la clase Console para leer datos introducidos desde el teclado o para escribir datos en la pantalla.

- Permite acceder a las secuencias estándar de entrada, salida y error
- Sólo tiene sentido para aplicaciones de consola
  - Entrada estándar: teclado
  - Salida estándar: Pantalla
  - Error estándar: Pantalla
- Es posible redireccionar todas las secuencias

Recomendación al profesor
Insista en que la clase Console se debe utilizar únicamente para aplicaciones de línea de comandos. Los desarrolladores que escriban aplicaciones de interfaz gráfica de usuario (GUI) tienen que usar clases del espacio de nombres System.Windows.Forms.

La clase **Console** hace que una aplicación C# pueda acceder a las secuencias estándar de entrada, salida y error.

La entrada estándar está asociada normalmente con el teclado, de forma que todo lo que el usuario escribe en el teclado se puede leer desde la secuencia de entrada estándar. Del mismo modo, la secuencia de salida estándar suele estar dirigida a la pantalla, al igual que la secuencia de error estándar.

Nota Estas secuencias y la clase **Console** sólo tienen sentido para aplicaciones de consola, que son las que se ejecutan en una ventana Command (Comandos).

Es posible direccionar cualquiera de las tres secuencias (entrada estándar, salida estándar, error estándar) a un archivo o dispositivo, tanto durante la programación como al ejecutar la aplicación.

Los métodos Write y WriteLine

Objetivo del tema
Explicar los métodos **Write** y **WriteLine** de la clase **Console**.

Explicación previa
Los métodos **Write** y **WriteLine** permiten enviar información a la secuencia de salida estándar.

- Console.Write y Console.WriteLine muestran información en la pantalla de la consola
  - WriteLine envía un fin de línea/retorno de carro
- Ambos métodos son sobrecargados
- Es posible emplear una cadena de formato y parámetros
  - Formatos de texto
  - Formatos numéricos

Recomendación al profesor
Puede ser conveniente buscar paralelismos con **cout** en C++, **printf** en C y **System.out.print** en Java, dependiendo de los estudiantes. En Visual Basic no hay ninguna función que sea realmente equivalente.

Procure también que la discusión sobre sobrecarga de métodos sea corta y simple. Es posible que algunos estudiantes aún no la entiendan, y además se tratará con más detalle en un módulo posterior.

Los métodos **Console.Write** y **Console.WriteLine** se pueden utilizar para mostrar información en la pantalla de la consola. Los dos métodos son muy similares; la diferencia más importante es que **WriteLine** añade un fin de línea/retorno de carro al final de la salida, mientras que **Write** no lo hace.

Ambos métodos son sobrecargados. Puede explicarlo con números variables y tipos de parámetros. Por ejemplo, puede usar el siguiente código para escribir “99” en la pantalla:

```
Console.WriteLine(99);
```

También puede usar el siguiente código para escribir “Hola, mundo” en la pantalla:

```
Console.WriteLine("Hola, mundo");
```

Formatos de texto

Existen formas de **Write** y **WriteLine** más potentes, que toman una cadena de formato y parámetros adicionales. La cadena de formato especifica cómo aparecen los datos y puede contener marcadores, que son sustituidos por los parámetros que siguen. Por ejemplo, puede usar el siguiente código para mostrar el mensaje “La suma de 100 y 130 es 230”:

```
Console.WriteLine("La suma de {0} y {1} es {2}", 100, 130,
100+130);
```

Importante El primer parámetro que sigue a la cadena de formato se referencia como parámetro cero: {0}.

Se puede utilizar el parámetro de la cadena de formato para especificar anchuras de campos y para indicar si los valores en esos campos deben ir justificados a derecha o a izquierda, como se ve en el siguiente código:

```
Console.WriteLine("\"Justificación a la izquierda en un campo
de anchura 10: {0, -10}\"", 99);
Console.WriteLine("\"Justificación a la derecha en un campo de
anchura 10: {0,10}\"", 99);
```

Esto hará que en la consola aparezca lo siguiente:

“Justificación a la izquierda en un campo de anchura 10: 99 ”

“Justificación a la derecha en un campo de anchura 10: 99”

Nota Para desactivar el significado especial de un carácter en una cadena de formato se puede usar una barra diagonal inversa (\) antes de ese carácter. Por ejemplo, "\{" hará que aparezca un "{" literal y "\\" mostrará un "\" literal. También se puede emplear el carácter @ para representar al pie de la letra una cadena entera. Por ejemplo, @"\\server\share" dará como resultado "\\server\share."

Formatos numéricos

Es posible utilizar la cadena de formato para especificar el formato de datos numéricos. La sintaxis completa para la cadena de formato es `{N,M:FormatString}`, donde `N` es el número del parámetro, `M` es la anchura y justificación del campo, y `FormatString` indica cómo se deben mostrar los datos numéricos. La tabla siguiente resume los valores que puede adoptar **FormatString**. Opcionalmente, en todos estos formatos se puede especificar el número de dígitos que se desea mostrar o al que se debe redondear.

Valor	Significado
C	Muestra el número como una unidad monetaria, usando el símbolo y las convenciones de la moneda local.
D	Muestra el número como un entero decimal.
E	Muestra el número como usando notación exponencial (científica).
F	Muestra el número como un valor en coma fija.
G	Muestra el número como un valor entero o en coma fija, dependiendo del formato que sea más compacto.
N	Muestra el número con comas incorporadas.
X	Muestra el número utilizando notación hexadecimal.

El siguiente código muestra algunos ejemplos de formatos numéricos:

```
Console.WriteLine("Formato de moneda - {0:C}  {1:C4}", 88.8,
➥-888.8);
Console.WriteLine("Formato entero - {0:D5}", 88);
Console.WriteLine("Formado exponencial - {0:E}", 888.8);
Console.WriteLine("Formato de punto fijo - {0:F3}",
➥888.8888);
Console.WriteLine("Formato general - {0:G}", 888.8888);
Console.WriteLine("Formato de número - {0:N}", 8888888.8);
Console.WriteLine("Formato hexadecimal - {0:X4}", 88);
```

El resultado de ejecutar este código es el siguiente:

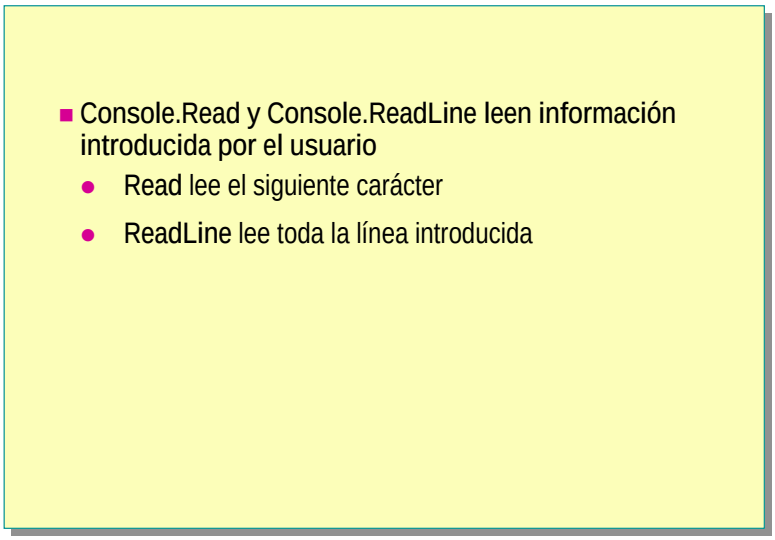
```
Formato de moneda - $88.80  ($888.8000)
Formato entero - 00088
Formado exponencial - 8.888000E+002
Formato de punto fijo - 888.889
Formato general - 888.8888
Formato de número - 8,888,888.80
Formato hexadecimal – 0058
```

Nota Para más información sobre formatos, busque “cadenas de formato” en la ayuda de Microsoft MSDN®.

Los métodos Read y ReadLine

Objetivo del tema
Explicar los métodos Read y ReadLine de la clase Console.

Explicación previa
Los métodos Read y ReadLine permiten obtener información introducida por el usuario con el teclado.



Recomendación al profesor
No se ofrece ningún ejemplo de Read porque, para tener algún sentido, el ejemplo tendría que mostrar cómo pasar de un entero (devuelto por Read) a un carácter:

```
int i;
char c;
i = Console.Read( );
c = (char) i;
Console.WriteLine(c);
```

Los métodos **Console.Read** y **Console.ReadLine** se pueden utilizar para mostrar información introducida por el usuario con el teclado.

El método Read

Read lee el siguiente carácter desde el teclado. Devuelve el valor **int** –1 si ya no hay más información. De lo contrario, devuelve un **int** que representa el carácter leído.

El método ReadLine

ReadLine lee todos los caracteres hasta el final de la línea introducida (el retorno de carácter de carro). La información introducida se devuelve como una cadena de caracteres. El siguiente código se puede utilizar para leer una línea de texto desde el teclado y mostrarla en la pantalla:

```
string input = Console.ReadLine( );
Console.WriteLine("{0}", input);
```

Comentarios a aplicaciones

Objetivo del tema
Explicar cómo hacer comentarios a aplicaciones.

Explicación previa
Todas las aplicaciones deben llevar comentarios adecuados.

- Los comentarios son importantes
  - Una aplicación con los comentarios adecuados permite a un desarrollador comprender perfectamente la estructura de la aplicación
- Comentarios de una sola línea

```
// Obtener el nombre del usuario
Console.WriteLine("¿Cómo se llama? ");
name = Console.ReadLine( );
```

- Comentarios de varias líneas

```
/* Encontrar la  mayor raíz de la
   ecuación cuadrática */
x = (…);
```

Recomendación al profesor
Los comentarios son importantes, pero su distribución y densidad dependen de las preferencias del desarrollador. Pregunte a los estudiantes cuáles son las normas sobre comentarios seguidas en sus organizaciones.

Es importante que todas las aplicaciones tengan una documentación adecuada. Siempre hay que incluir suficientes comentarios para que un desarrollador que no participara en la creación de la aplicación original pueda seguir y comprender su funcionamiento. Los comentarios deben ser exhaustivos y pertinentes. Unos buenos comentarios añaden información que no es fácil de expresar usando sólo las instrucciones del código, ya que explican el “porqué” en lugar del “qué.” Siga las normas de su organización para comentar código (si las tiene).

C# ofrece varios mecanismos que permiten añadir comentarios al código de aplicaciones: comentarios de una sola línea, comentarios de varias líneas y documentación generada en XML.

Para añadir un comentario de una sola línea se pueden utilizar los caracteres de barra diagonal (//). Al ejecutar la aplicación, se ignora todo lo que sigue a estos dos caracteres hasta el final de la línea.

También es posible hacer comentarios en bloques de varias líneas. Un comentario en bloque empieza con el par de caracteres /* y continúa hasta llegar al el par de caracteres */ correspondiente. Los comentarios en bloques no pueden estar anidados.

Tratamiento de excepciones

Objetivo del tema
Explicar cómo capturar excepciones de tiempo de ejecución en C#.

Explicación previa
Una buena aplicación C# debe tener capacidad para enfrentarse a lo inesperado.

```
using System;
public class Hello
{
  public static void  Main(string [ ] args)
  {
     try{
         Console.WriteLine(args[0]);
        }
     catch (Exception e) {
         Console.WriteLine("Excepción en
        ➥{0}", e.StackTrace );
     }
  }
}
```

Recomendación al profesor
En este tema se ofrecen comentarios generales sobre el tratamiento de excepciones. Se darán más detalles en el módulo 4, "Instrucciones y excepciones", del curso 2124C, *Programación en C#*.

Una buena aplicación C# debe tener capacidad para enfrentarse a lo inesperado. Por muchas comprobaciones de error que se añadan al código, siempre habrá algo que pueda ir mal. El usuario puede dar una respuesta imprevista a una pregunta, por ejemplo, o tratar de escribir a un archivo en una carpeta que ha sido borrada. Las posibilidades son infinitas.

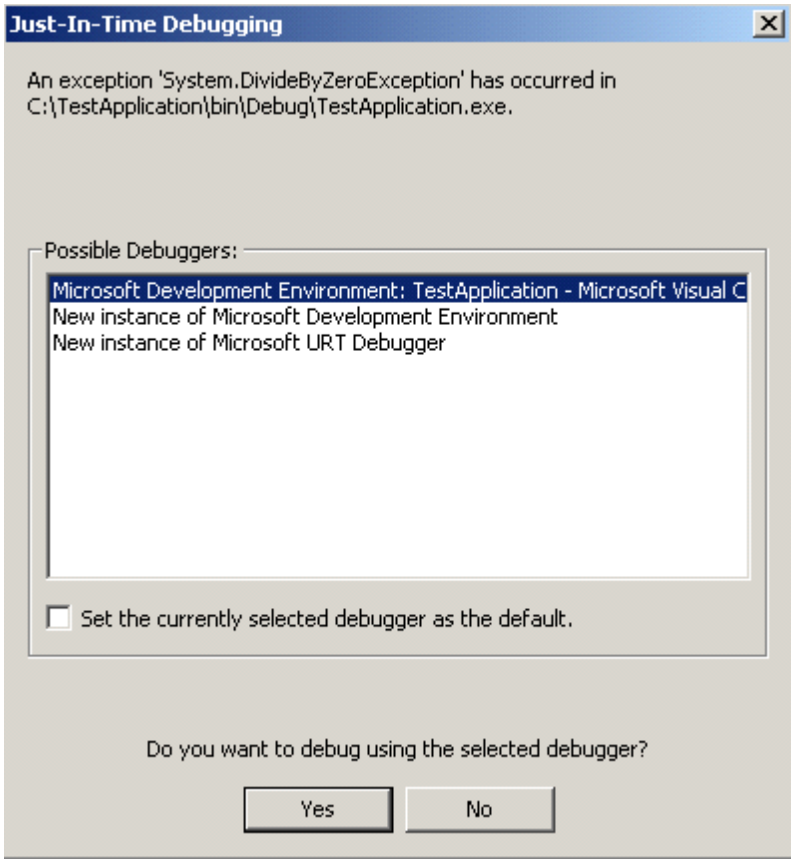
Si se produce un error de tiempo de ejecución en una aplicación C#, el sistema operativo lanza una excepción. Las excepciones se pueden capturar con una combinación de las instrucciones **try** y **catch**, como se muestra en la transparencia. Si alguna de las instrucciones en la parte **try** de la aplicación hace que se produzca una excepción, la ejecución pasará al bloque **catch**.

Para obtener información sobre la excepción producida se pueden emplear las propiedades **StackTrace**, **Message** y **Source** del objeto **Exception**. El tratamiento de excepciones se tratará con más detalle en un módulo posterior.

Nota Si se imprime una excepción, utilizando por ejemplo **Console.WriteLine**, la excepción se dará formato automáticamente y mostrará las propiedades **StackTrace**, **Message** y **Source**.

Consejo Incluir el tratamiento de excepciones en aplicaciones C# desde el principio es mucho más sencillo que tratar de añadirlo después.

Si no se utiliza tratamiento de excepciones se producirá una excepción en tiempo de ejecución. Si en lugar de ello se prefiere depurar un programa utilizando depuración Just-in-Time, es preciso activarla antes. Una vez activada, depuración Just-in-Time indicará el depurador que hay que utilizar dependiendo del entorno y de las herramientas instaladas.



Ejecute los siguientes pasos para activar la depuración Just-in-Time:

1. En el menú **Tools** (Herramientas), pulse **Options** (Opciones).
2. En el cuadro de diálogo **Options**, haga clic en la carpeta **Debugging** (Depuración).
3. En la carpeta **Debugging**, pulse **Just-In-Time Debugging** (Depuración Just-in-Time).
4. Active o desactive la depuración Just-in-Time (JIT) para distintos tipos de programas y pulse **OK**.

El depurador se estudiará en detalle más adelante en este módulo.

◆ Compilación, ejecución y depuración

Objetivo del tema
Ofrecer una introducción a los temas tratados en esta sección.

Explicación previa
En esta sección aprenderá a compilar, ejecutar y depurar programas C#.

- Llamadas al compilador
- Ejecución de la aplicación
- Demostración: Compilación y ejecución de un programa C#
- Depuración
- Demostración: Uso del depurador de Visual Studio
- Las herramientas del SDK
- Demostración: Uso del ILDASM

En esta lección aprenderá a compilar y depurar programas C#. Verá cómo se ejecuta el compilador desde la línea de comandos y desde el entorno Visual Studio. Aprenderá algunas opciones comunes de compilador y conocerá el Visual Studio Debugger. Finalmente, aprenderá a utilizar algunas de las otras herramientas del kit de desarrollo de software (SDK) de Microsoft .NET Framework..

Llamadas al compilador

Objetivo del tema
Explicar cómo compilar una aplicación C# y las diferentes opciones de compilador disponibles.

Explicación previa
En preciso compilar las aplicaciones C# antes de ejecutarlas.

- Conmutadores comunes del compilador
- Compilación desde la línea de comandos
- Compilación desde Visual Studio
- Localización de errores

Antes de ejecutar una aplicación C# es necesario compilarla. El compilador convierte el código fuente escrito por usted en código máquina que puede entender el sistema. Se pueden hacer llamadas al compilador de C# desde la línea de comandos o desde Visual Studio.

Nota Estrictamente hablando, las aplicaciones C# se compilan a lenguaje intermedio de Microsoft (MSIL) y no al código máquina nativo. El código MSIL se compila a su vez a código máquina con el compilador Just-in-Time (JIT) cuando se ejecuta la aplicación. No obstante, también es posible compilar directamente a código máquina y evitar el compilador JIT empleando la utilidad Native Image Generator (Ngen.exe), que crea una imagen nativa a partir de un ensamblado administrado y la instala en la caché de imagen nativa del sistema local. La ejecución de Ngen.exe sobre un ensamblado hace que éste se cargue más rápido, ya que restaura código y estructuras de datos desde la caché de imagen nativa en lugar de generarlas de forma dinámica.

Conmutadores comunes del compilador

Es posible especificar distintos conmutadores para el compilador de C# usando el comando **csc**. La siguiente tabla describe los conmutadores más comunes.

Conmutador	Significado
/?, /help	Muestra las opciones del compilador en la salida estándar.
/out	Especifica el nombre del ejecutable.
/main	Especifica la clase que contiene el método **Main** (si en la aplicación hay más de una clase que incluya un método **Main**).
/optimize	Activa y desactiva el optimizador de código.

(*continúa*)

Conmutador	Significado
/warn	Fija el nivel de aviso del compilador.
/warnaserror	Trata todos los avisos como errores que interrumpen la compilación.
/target	Especifica el tipo de aplicación generada.
/checked	Indica si un desbordamiento aritmético genera una excepción en tiempo de ejecución.
/doc	Procesa comentarios de documentación para crear un archivo XML.
/debug	Genera información sobre la depuración.

Compilación desde la línea de comandos

Para compilar una aplicación C# desde la línea de comandos se emplea el comando **csc**. Por ejemplo, para compilar la aplicación Hola, mundo (Hola.cs) desde la línea de comandos, generar información sobre la depuración y crear un ejecutable llamado Saludo.exe, el comando es:

```
csc /debug+ /out:Saludo.exe Hola.cs
```

Importante El archivo de salida que contiene el código compilado debe incluir el sufijo .exe. Si se omite el sufijo, es necesario cambiar el nombre del archivo antes de ejecutarlo.

Compilación desde Visual Studio

Para compilar una aplicación C# utilizando Visual Studio, abra el proyecto que contiene la aplicación C# y seleccione **Build Solution** (Compilar solución) en el menú **Build**.

Nota Por defecto, Visual Studio abre la configuración de depuración (Debug) para el proyecto. Esto significa que lo que se compila es una versión de depuración de la aplicación. Si desea compilar una versión de lanzamiento que no contenga información sobre la depuración, cambie la configuración de la solución a lanzamiento (Release).

Para cambiar las opciones empleadas por el compilador se puede modificar la configuración del proyecto:

1. En Solution Explorer, pulse con el botón derecho del ratón en el icono del proyecto.
2. Pulse **Properties** (Propiedades).
3. En el cuadro de diálogo **Property Pages** (Páginas de propiedades), pulse **Configuration Properties** (Propiedades de configuración) seguido de **Build**.
4. Especifique las opciones que desee del compilador y pulse **OK**.

Localización de errores

El compilador de C# informará de los errores sintácticos o semánticos que detecte.

Si la llamada al compilador se hizo desde la línea de comandos, el compilador mostrará mensajes en los que indicará los números de línea y el carácter dentro de cada línea donde se encontraran errores.

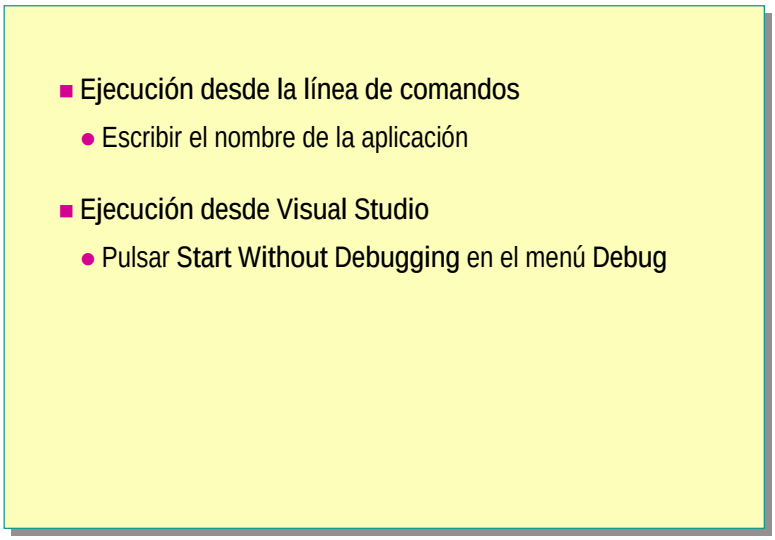
Si la llamada al compilador se hizo desde Visual Studio, la ventana Task List (Lista de tareas) mostrará todas las líneas que incluyan errores. Haga doble clic sobre una línea de esta ventana para ir al error correspondiente en la aplicación.

Consejo Es frecuente que un solo error de programación varios errores de compilación. Lo mejor es repasar todos los errores a partir de los que se encontraron primero, ya que corregir un error al principio del código puede hacer que desaparezcan automáticamente varios errores posteriores.

Ejecución de la aplicación

Objetivo del tema
Explicar cómo se ejecuta una aplicación compilada.

Explicación previa
Una aplicación C# se puede ejecutar desde la línea de comandos o desde Visual Studio.



Una aplicación C# se puede ejecutar desde la línea de comandos o desde el entorno Visual Studio.

Ejecución desde la línea de comandos

Cuando la aplicación se compila sin problemas se genera un archivo ejecutable (un archivo cuyo sufijo es .exe). Para ejecutarlo desde la línea de comandos, escriba el nombre de la aplicación (con o sin el sufijo .exe).

Ejecución desde Visual Studio

Para ejecutar la aplicación desde Visual Studio, pulse **Start Without Debugging** (Iniciar sin depuración) en el menú **Debug** (Depurar) o pulse CTRL+F5. Si se trata de una aplicación de consola, se abrirá inmediatamente una ventana de consola y la aplicación se ejecutará. Una vez finalizada la aplicación, el sistema le pedirá que pulse cualquier tecla para continuar y la ventana de consola se cerrará.

Depuración

Objetivo del tema
Explicar el uso de Visual Studio para depurar una aplicación.

Explicación previa
Visual Studio contiene muchas herramientas prácticas que le permitirán depurar y corregir aplicaciones.

- Excepciones y depuración JIT
- El Visual Studio Debugger
  - Configuración de puntos de interrupción e inspecciones
  - Seguimiento del código paso a paso
  - Examen y modificación de variables

Excepciones y depuración JIT

Si una aplicación lanza una excepción y no se ha escrito el código necesario para ese caso, el runtime de lenguaje común hará que se inicie una depuración JIT (no hay que confundir la depuración JIT con el compilador JIT).

Suponiendo que se ha instalado Visual Studio, se abrirá un cuadro de diálogo que dará a elegir entre depurar la aplicación con el Visual Studio Debugger (entorno de desarrollo de Microsoft) o con el depurador del SDK de .NET Framework.

Si se dispone de Visual Studio, lo más recomendable es elegir el depurador del entorno de desarrollo de Microsoft.

Nota El SDK de .NET Framework ofrece otro depurador llamado cordbg.exe, que es un depurador para línea de comandos. Incluye la mayor parte de las funciones ofrecidas por el entorno de desarrollo de Microsoft, salvo la interfaz gráfica de usuario. En este curso no se discutirá este depurador.

Recomendación al profesor
Abra Visual Studio y muestre a los estudiantes las opciones de menú y las barras de herramientas durante la discusión.

Después de este tema encontrará una demostración más completa.

Configuración de puntos de interrupción e inspecciones en Visual Studio

Se puede utilizar el Visual Studio Debugger para fijar puntos de interrupción en el código y examinar los valores de variables.

Al pulsar con el botón derecho del ratón sobre una línea de código se abre un menú con muchas opciones útiles. Seleccione **Insert Breakpoint** para insertar un punto de interrupción en esa línea. También puede insertar un punto de interrupción pulsando en el margen izquierdo. Vuelva a pulsar para borrar el punto. Al ejecutar la aplicación en modo de depuración, la ejecución se detendrá en esta línea y será posible examinar el contenido de variables.

La ventana Watch (Inspección) resulta útil para supervisar los valores de variables seleccionadas mientras se ejecuta la aplicación. Si escribe el nombre de una variable en la columna **Name** (Nombre), su valor aparecerá en la columna **Value** (Valor), por lo que podrá ver todos los cambios que se produzcan durante la ejecución. También es posible sobrescribir el valor de una variable inspeccionada para modificarla.

Importante Antes de utilizar el depurador, asegúrese de que la configuración de la solución es Debug y no Release.

Seguimiento del código paso a paso

Una vez fijados los puntos de interrupción necesarios, se puede ejecutar la aplicación seleccionando **Start** (Inicio) en el menú **Debug** o pulsando F5. La ejecución se detendrá al llegar al primer punto de interrupción.

La ejecución se reanudará al seleccionar **Continue** (Continuar) en el menú **Debug**, aunque también se pueden emplear las opciones de avance paso a paso del menú **Debug** para avanzar en el código línea por línea.

Consejo Las opciones de punto de interrupción, avance paso a paso e inspección de variables se encuentran también en la barra de herramientas **Debug**.

Examen y modificación de variables

Para ver las variables definidas en un método se puede pulsar **Locals** (Locales) en la barra de herramientas **Debug** o bien utilizar la ventana Watch. También es posible sobrescribir el valor de una variable para modificarla (como en la ventana Watch).

Uso del Visual Studio Debugger

Este programa C#, que convierte temperaturas de Fahrenheit a Celsius, contiene un error que en algunos casos produce resultados incorrectos en la conversión de temperaturas. Encuentre y corrija el problema.

Cómo ejecutar el programa C#

1. Abra el proyecto Converter, que se encuentra en la carpeta Converter dentro del archivo demo02.zip.
2. Ejecute el programa seleccionando **Start Without Debugging** en el menú **Debug** o pulsando CTRL+F5.

   Explique que el programa pide al usuario una temperatura en grados Fahrenheit. A continuación convierte ese valor en un entero y lo almacena en la variable *degreesFahrenheit*. Luego calcula la temperatura Celsius equivalente con la fórmula de conversión estándar y almacena el resultado en la variable *degreesCelsius*. Finalmente, el programa muestra la temperatura Fahrenheit original junto con el valor calculado en Celsius.
3. Escriba **32** cuando el sistema le pida una temperatura.

   Haga notar a los estudiantes que programa convierte esta temperatura en 0 grados Celsius, que es correcto.
4. Vuelva a ejecutar el programa y escriba **212** cuando el sistema le pida un valor. Esta temperatura es el punto de ebullición del agua.

   Haga notar a los estudiantes que el programa da un resultado de 100 grados Celsius, que también es correcto.
5. Ejecute el programa por tercera vez y escriba una temperatura de **75** grados Fahrenheit.

   Haga notar a los estudiantes que el resultado mostrado es 23 grados Celsius. Esto es incorrecto, ya que 75 grados Fahrenheit son 23,8889 grados Celsius. Explique que, aparentemente, el programa está truncando el resultado a un entero.

Cómo identificar el error en el programa C# estableciendo un punto de interrupción y usando la ventana Watch (Inspección)

1. Coloque un punto de interrupción justo después de que programa lee la temperatura Fahrenheit.
2. Ejecute el programa en modo de depuración seleccionando **Start** en el menú **Debug** o pulsando.

   Escriba **75** y pulse INTRO. El programa llega hasta el punto de interrupción y se detiene.
3. Abra la ventana Watch y añada las variables *degreesFahrenheit* y *degreesCelsius* a la lista de variables inspeccionadas.

   Explique que es posible supervisar los valores de variables en la ventana Watch.

4. Seleccione **Step Over** (Salto) en el menú **Debug** o pulse F10 para ejecutar el programa línea por línea a partir del punto de interrupción.

   En esto consiste el avance paso a paso. Se ejecuta la siguiente línea de código y el programa se detiene, lo que permite ver en la ventana Watch los cambios sufridos por las variables. Por ejemplo, el valor de *degreesFahrenheit* es ahora 75. Al saltar a la siguiente línea, la variable *degreesCelsius* cambia cuando se calcula un valor de 23.0.

   Ésta es la línea que contiene el error. El problema es que se realiza un cálculo entero sobre un dato entero antes de asignarlo a una variable de punto flotante. Así se acumulan errores de redondeo y se obtiene un resultado incorrecto.

5. Pulse F5 para finalizar la ejecución del programa.

Cómo corregir el error en el programa C# y comprobar el resultado

1. Cambie una constante a un valor de punto flotante cambiando 32 a 32F, lo que obliga al compilador a efectuar un cálculo de punto flotante.
2. Vuelva a compilar y ejecutar el programa.
3. Seleccione **Step Over** en el menú **Debug** para ver los resultados del cambio.

   Verá que ahora la variable *degreesCelsius* tiene un valor de 23.8889 en la ventana Watch.

4. Pulse F10 para continuar ejecutando el programa paso a paso.
5. Seleccione **Continue** en el menú **Debug** para que el programa se ejecute hasta el final.