Contenidos

Tecnologías de Desarrollo de Software IDE

Unidad: 3

“Sintaxis y Orientación a Objetos de Lenguajes .Net”

(Última Actualización: 10 Abril 2012)

**Índice**

[Unidad 03: Sintaxis y Orientación a Objetos de Lenguajes .Net](#h.8qrkdx2nclre)

[Introducción](#h.nwkf9yjbixn8)

[Sintaxis de Lenguajes .Net](#h.17r6oypdi8hd)

[Lenguaje C#](#h.ql3g85yc4j4o)

[Excepciones y Errores](#h.fq61878slotr)

[Ventajas](#h.8u4a0hrp8spo)

[Cómo: Detectar y manejar excepciones utilizando el bloque Try/Catch](#h.bul8v44iw6hz)

[Cómo: Utilizar excepciones específicas en un bloque Catch](#h.fffrni9ac14r)

[Cómo: Iniciar excepciones explícitamente](#h.261natp046x0)

[Cómo: Utilizar controladores de excepciones filtrados por el usuario](#h.rrhtovbrpmst)

[Cómo: Utilizar bloques Finally](#h.9ps6t9pvj2ki)

[Cómo: Crear clases personalizadas de Excepciones](#h.xvu4c2dix609)

[Modificadores de Acceso](#h.26v30p445l15)

[Namespaces (Espacios de Nombres)](#h.o9l1s7e7k2r9)

[Palabra reservada using](#h.9hngop12lkw6)

[Enumeraciones](#h.lp72bdybfcyk)

[Nota IMPORTANTE: este tema no esta desarrollado aún.](#h.slmddxun0ob9)

[Consultar entre otras fuentes en MSDN “La clase Enum”](#h.iqceeyspgh2t)

[Orientación a Objetos](#h.83rrv4phyduy)

[Ejemplos de Código](#h.r74b7um9wlvv)

[Resumen de la Unidad / Capítulo](#h.1e22yl4cuw8g)

[Bibliografía#](#h.fr1y1imghqss)

[Anexo A: Equivalencias y diferencias entre lenguajes .Net](#h.8ab12jffm03)

## 

## Unidad 03: Sintaxis y Orientación a Objetos de Lenguajes .Net

**IMPORTANTE:** los **contenidos** aquí desarrollados **no son necesariamente completos** en su total profundidad, por lo que **se sugiere consultar otras fuentes**, tales como la **Bibliografia** sugerida por la cátedra en el Programa Analítico y docentes.

Específicamente en este documento NO se encuentran aun desarrollados todos los temas planeados para esta Unidad según Programa Analítico vigente.

**Objetivo(s)**

* Conocer fundamentos básicos y sintaxis del lenguaje de programación C# .Net y Visual Basic .Net.
* Aplicar conceptos de Programación Orientada a Objetos (POO), aprendidos en otras asignaturas, en .Net.

**Temas**

1. Tipos, variables y constantes.
2. Declaración, Asignación e Inicialización.
3. Alcance y Visibilidad.
4. Operadores y palabras reservadas.
5. Conversión de Tipos
6. Estructuras de Control: Decisión y Iteración
7. Manejo de Errores y Excepciones.
8. Uso y creación de Clases y Objetos en .Net
9. Abstracción y Encapsulamiento.
10. Herencia.
11. Polimorfismo.
12. Conceptos de POO aplicados a C# y diferencias con VB.Net.
13. Constructores y destructores.
14. Propiedades y Enumeradores (Enums)
15. Métodos y atributos.
16. Herencia e interfaces.
17. Ocultamiento.
18. Clases abstractas y métodos virtuales.
19. Clases parciales.
20. Modificadores de acceso/alcance, visibilidad y herencia.
21. Diseñador de clases.
22. Espacio de Nombres (Namespaces)

### Introducción

La plataforma .Net soporta[[1]](#footnote-0) una gran variedad de lenguajes, dentro de los que se encuentran los desarrollados por el propio Microsoft, tales como C#[[2]](#footnote-1), Visual Basic .NET, C++, J# y recientemente F#[[3]](#footnote-2) (lenguaje funcional) o los desarrollados por otros fabricantes tales como, Cobol, Phyton, Delphi (Object Pascal), Perl, Python, Fortran, Prolog y PowerBuilder entre otros.

Los lenguajes de programación de la plataforma .NET, utilizan los servicios y características de .NET Framework a través de un conjunto común de clases unificadas, la BCL.

En la mayoría de las situaciones, puede utilizar de manera eficiente todos los lenguajes de programación. Sin embargo, cada lenguaje tiene sus puntos fuertes, y es recomendable comprender las características únicas para cada uno de ellos.

La elección de un lenguaje de programación depende de sus conocimientos del lenguaje y del ámbito de la aplicación que está generando. Las aplicaciones de pequeño tamaño se suelen crear utilizando un único lenguaje, y en otros casos pueden utilizarse mas de uno de acuerdo a los requerimientos particulares de la solución.

Existen diferentes fuentes que pueden ser consultadas, donde se tratan

**IMPORTANTE**: Se sugiere la búsqueda de bibliografia disponible en Biblioteca UTN Rosario pudiendo buscar títulos en el siguiente buscador con algunas palabras claves sugeridas.

Buscador **Biblioteca UTN**: <http://www.frro.utn.edu.ar/biblioteca.php>

Posible **palabras claves** para realizar la búsqueda: **.Net**, **C#**, **Visual Basic .Net**, **Objetos**, **Object**.

### **Sintaxis de Lenguajes .Net**

Se plantean diferentes aspectos de la sintaxis de diferentes lenguajes soportados por la plataforma .Net, y en especial se profundizara sobre el lenguaje C# presentando comparativas con otros lenguajes soportados tales como Visual Basic. Net, tanto a nivel conceptual como en el planteo de ejemplos de código, mostrando diferencias, similitudes y equivalencias entre ellos.

#### Lenguaje C#

El lenguaje de programación **C#**, pronunciado C Sharp, es en la actualidad junto a Java, uno de los mas populares, que permite tanto el desarrollo de aplicaciones para Internet como de aquellas de propósito general.

**MUY IMPORTANTE**: dado que los contenidos incluidos en este material no abarcan los planteados para la unidad por el momento, se solicita remitirse al resto de material que la cátedra dejo a disposicion sumado a otro material al que tengan acceso (para el cual se sugiere consultar con los docentes para recibir sugerencias sobre el miso).

### Excepciones y Errores

Al escribir código es posible que se introduzcan errores, algunos de los cuales pueden ser detectados en tiempo de compilación, tales como los de sintaxis o semántica; mientas que en otros casos pueden pasar inadvertidos por ser errores de lógica o requerir ciertas condiciones para manifestarse.

Es posible que la ejecución de un programa no se realice de la forma esperada y se generen errores al presentarse alguna situación imprevista que impida que la aplicación siga corriendo.

Podríamos considerar situaciones como por ejemplo, el ingreso de una entrada incorrecta de datos, una división por cero, un formato de salida no valido, una falla en la conexión a una fuente de datos o alguna otra situación inesperada.

Los programas deben poder controlar los errores que se producen durante la ejecución de manera uniforme sin que la aplicación deje de funcionar[[4]](#footnote-3). Para ello el Framework .Net contiene un sistema de **manejo de excepciones[[5]](#footnote-4)** muy potente, que permiten lidiar con situaciones no previstas de modo muy sencillo y efectivo. Este sistema cuenta con un conjunto de clases (incluidas en la BCL), con métodos y propiedades que permiten el manejo de situaciones no esperadas.

Una **excepción** es una alteración del flujo normal de ejecución de una aplicación.

Al generarse una situación no esperada, el **.NET Framework lanza una excepción**, para informar del error (o anomalía), la cual será necesario capturar y poder gestionar.

Anteriormente, el modelo de control de errores de un lenguaje dependía de la forma exclusiva que tenía el lenguaje de detectar los errores y buscarles controladores o del mecanismo de control de errores proporcionado por el sistema operativo.

Ahora el motor en tiempo de ejecución (runtime)[[6]](#footnote-5) implementa el control de excepciones con las características siguientes:

* Controla las excepciones independientemente del lenguaje que las genera o controla.
* No requiere una sintaxis concreta del lenguaje para controlar las excepciones, aunque permite a cada lenguaje definir la propia.
* Permite que las excepciones se inicien traspasando los límites de los procesos e, incluso, los equipos.

#### Ventajas

Las excepciones ofrecen varias ventajas respecto a otros métodos de notificación de error, tal como aquellos que devuelven códigos de error, entre los que podemos citar:

* Ningún error pasa desapercibido.
* Los valores no válidos no se siguen propagando por el sistema.
* No es necesario comprobar los códigos devueltos.
* Es muy sencillo agregar código de control de excepciones para aumentar la confiabilidad del programa.

Las excepciones predefinidas en la plataforma .Net, son subclases derivadas de la clase base **System.Exception[[7]](#footnote-6)** que abarcan diferentes tipo de excepciones tales como, las aritméticas, de formato, acceso al índice de una posición de un arreglo fuera del limite, división por cero, entre otras. Además se pueden extender el abanico ofrecido por el sistema, creando nuevas derivadas de alguna de ellas por medio de herencia.

En esta sección veremos:

[Cómo: Utilizar el bloque Try/Catch para detectar excepciones](http://msdn.microsoft.com/es-ar/library/xtd0s8kd(v=vs.80).aspx)

Describe cómo usar el bloque try/catch para controlar excepciones.

[Cómo: Utilizar excepciones específicas en un bloque Catch](http://msdn.microsoft.com/es-ar/library/3tca6706(v=vs.80).aspx)

Describe cómo detectar excepciones específicas.

[Cómo: Iniciar una excepción explícitamente](http://msdn.microsoft.com/es-ar/library/xhcbs8fz(v=vs.80).aspx)

Describe cómo producir excepciones y cómo detectarlas y, a continuación, volver a producirlas otra vez.

[Cómo: Crear excepciones definidas por el usuario](http://msdn.microsoft.com/es-ar/library/87cdya3t(v=vs.80).aspx)

Describe cómo crear clases de excepciones personalizadas.

[Utilizar controladores filtrados por el usuario](http://msdn.microsoft.com/es-ar/library/4dy8x9k9(v=vs.80).aspx)

Describe cómo configurar excepciones filtradas.

[Cómo: Utilizar un bloque Finally](http://msdn.microsoft.com/es-ar/library/ke0zf0f5(v=vs.80).aspx)

Explica cómo utilizar la instrucción Finally en un bloque de excepciones.

Cómo: Crear clases personalizadas de Excepciones

Describe cómo crear excepciones personalizadas.

Secciones relacionadas

[Información general sobre excepciones](http://msdn.microsoft.com/es-ar/library/6kzk0czb(v=vs.80).aspx)

Proporciona una descripción general de las excepciones de Common Language Runtime.

[Clase Exception y propiedades](http://msdn.microsoft.com/es-ar/library/5whzhsd2(v=vs.80).aspx)

Describe los elementos de un objeto de excepción.

#### Cómo: Detectar y manejar excepciones utilizando el bloque Try/Catch

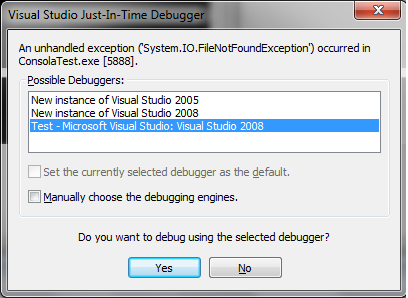
Para capturar una excepción se deben realizar minimamente dos cuestiones:

1. Colocar el código que pueda lanzar una excepción dentro de un **bloque Try**. En C# usando la palabra reservada **try**.
2. Colocar el código que maneja (controla) la excepción, cuando se lance, en un **bloque Catch**. En C# usando la palabra reservada **catch** seguida por un tipo de excepción y la acción que se debe tomar.

A modo de ejemplo supongamos que tenemos una aplicación de consola que tiene que recuperar datos de un archivo de texto, en el código aparece un método Main contiene un una instrucción StreamReader que abre un archivo de datos denominado data.txt y escribe una cadena del archivo.

|  |
| --- |
| // (C#) Manejo de excepción básico  using System;  using System.IO;  using System.Security.Permissions;  // Security permission request.  [assembly:FileIOPermissionAttribute(SecurityAction.RequestMinimum, All = @"c:\data.txt")] public class ProcessFile {  public static void Main() {  StreamReader sr = File.OpenText("data.txt");  Console.WriteLine("The first line of this file is {0}", sr.ReadLine());   } } |

Al ejecutarse el archivo y en caso que el archivo no este disponible, vamos a obtener un error en la aplicación que estando en desarrollo aparecerá una ventana de dialogo similar a la siguiente.



Ahora vamos a agregar un bloque try/catch para detectar una posible excepción. Colaremos las instrucciones para abrir el archivo y escribir una cadena del archivo dentro de un bloque **try**. Seguido agregaremos un bloque **catch** que detecta cualquier excepción que resulte del bloque Try quedando el código como se muestra seguido.

|  |
| --- |
| // (C#) Manejo de excepción básico  using System;  using System.IO;  using System.Security.Permissions;  // Security permission request.  [assembly:FileIOPermissionAttribute(SecurityAction.RequestMinimum, All = @"c:\data.txt")] public class ProcessFile {  public static void Main() {  try {  StreamReader sr = File.OpenText("data.txt");  Console.WriteLine("The first line of this file is {0}", sr.ReadLine());   }  catch(Exception e) {  Console.WriteLine("An error occurred: '{0}'", e);  }  } } |

De esta forma obtendremos la siguiente salida donde nos muestra el mensaje que agregamos mas el detalle de la excepción contenida en el objeto **e**.

|  |
| --- |
| An error occurred: 'System.IO.FileNotFoundException: Could not find file 'c:\data.txt'.  File name: 'c:\data.txt'  at System.IO.\_\_Error.WinIOError(Int32 errorCode, String maybeFullPath)  at System.IO.FileStream.Init(String path, FileMode mode, FileAccess access, I  nt32 rights, Boolean useRights, FileShare share, Int32 bufferSize, FileOptions o  ptions, SECURITY\_ATTRIBUTES secAttrs, String msgPath, Boolean bFromProxy)  at System.IO.FileStream..ctor(String path, FileMode mode, FileAccess access,  FileShare share, Int32 bufferSize, FileOptions options, String msgPath, Boolean  bFromProxy)  at System.IO.FileStream..ctor(String path, FileMode mode, FileAccess access,  FileShare share, Int32 bufferSize, FileOptions options)  at System.IO.StreamReader..ctor(String path, Encoding encoding, Boolean detec  tEncodingFromByteOrderMarks, Int32 bufferSize)  at System.IO.StreamReader..ctor(String path, Boolean detectEncodingFromByteOr  derMarks)  at System.IO.File.OpenText(String path)  at ProcessFile.Main() in C:\Temp\Net\Test\Test\ConsolaTest\Program.cs:line 36' |

Este ejemplo ilustra una instrucción Catch básica que detectará cualquier excepción. Por lo general, se recomienda, cuando se esté programando, detectar un tipo de excepción específico en lugar de utilizar la instrucción Catch básica. Para obtener información sobre la detección de excepciones específicas, vea [Utilizar excepciones específicas en un bloque Catch](http://msdn.microsoft.com/es-ar/library/3tca6706(v=vs.80).aspx).

#### Cómo: Utilizar excepciones específicas en un bloque Catch

Cuando se produce una excepción, asciende por la pila y se da a cada uno de los bloques Catch la oportunidad de controlarla. El orden de las instrucciones Catch es importante. Ponga los bloques Catch dirigidos a excepciones específicas antes de un bloque Catch de excepción general, pues de lo contrario el compilador puede emitir un error. El bloque Catch adecuado se determina haciendo coincidir el tipo de excepción con el nombre de la excepción especificada en el bloque Catch. Si no hay un bloque Catch específico, un bloque Catch general, si lo hay, detecta la excepción.

En el ejemplo de código siguiente se usa un bloque try/catch para detectar una InvalidCastException. En el ejemplo se crea una clase denominada Employee con una única propiedad, el nivel de empleado (Emlevel). Un método, PromoteEmployee, toma un objeto e incrementa el nivel del empleado. Se produce una InvalidCastException cuando se pasa una instancia de DateTime al método PromoteEmployee.

|  |
| --- |
| // (C#) Manejo de Excepciones Especificas  using System;  public class Employee  {  // Create employee level property.  int emlevel;  public int Emlevel  {  get { return(emlevel); }  set { emlevel = value; }  } }  public class Ex13  {  public static void PromoteEmployee(Object emp)  {  // Cast object to Employee.  Employee = (Employee) emp;  // Increment employee level.  e.Emlevel = e.Emlevel + 1;  }   public static void Main()  {  try  {  Object o = new Employee();  DateTime newyears = new DateTime(2001, 1, 1);  //Promote the new employee.  PromoteEmployee(o);  // Promote DateTime;  // results in InvalidCastException as newyears is not an employee instance.  PromoteEmployee(newyears);  }  catch (InvalidCastException e)  {  Console.WriteLine("Error passing data to PromoteEmployee method. " + e);  }  } } |

El Common Language Runtime detecta las excepciones que no detecta un bloque Catch. Según cómo esté configurado el motor en tiempo de ejecución, aparece un cuadro de diálogo de depuración o el programa se deja de ejecutar y aparece un cuadro de diálogo con información de la excepción. Para obtener información sobre la depuración, vea [Depurar y generar perfiles de aplicaciones](http://msdn.microsoft.com/es-ar/library/7fe0dd2y(v=vs.80).aspx).

#### Cómo: Iniciar excepciones explícitamente

Se puede producir una excepción de manera explícita utilizando la instrucción **Throw**. También se puede volver a producir una excepción detectada utilizando la instrucción Throw. Es muy recomendable, cuando se está escribiendo código, agregar información a una excepción que se vuelve a producir para proporcionar más información cuando se vaya a depurar.

En el ejemplo de código siguiente se usa un bloque try/catch para detectar una posible FileNotFoundException. Después del bloque Try hay un bloque Catch que detecta FileNotFoundException y escribe un mensaje en la consola si no se encuentra el archivo de datos. La siguiente instrucción es una instrucción Throw que produce una nueva FileNotFoundException y agrega información sobre el texto a la excepción.

|  |
| --- |
| // (C#) Manejo de excepción básico  using System;  using System.IO;  public class ProcessFile  {  public static void Main()  {  FileStream fs = null;  try  {  // Opens a text tile.  fs = new FileStream ("data.txt", FileMode.Open);  StreamReader sr = new StreamReader(fs);  string line;  // A value is read from the file and output to the console.  line = sr.ReadLine();  Console.WriteLine(line);   }  catch(FileNotFoundException e)  {  Console.WriteLine("[Data File Missing] {0}", e);  throw new FileNotFoundException("[data.txt not in c:\\dev directory]",e);  }  finally  {  fs.Close();  }  } } |

Notar que se incluye el bloque **Finaly** el cual va a ser ejecutado siempre independientemente de lo que pase en los bloques Try y Catch anteriores y en ese caso particular se agrega una sentencia para cerrar el archivo abierto dentro del bloque Try que permite liberar el recurso, una buena practica a tomar en cuenta para cuando se utilizan fuentes externas de datos.

#### **Cómo**: Utilizar controladores de excepciones filtrados por el usuario

Actualmente, Visual Basic admite las excepciones filtradas por el usuario. Los controladores de excepciones filtrados por el usuario detectan y controlan las excepciones basándose en requisitos que se definen para la excepción. Estos controladores utilizan la instrucción Catch con la palabra clave **When**.

Esta técnica resulta útil cuando un objeto de excepción concreto corresponde a varios errores. En este caso, normalmente, el objeto tiene una propiedad que contiene el código de error específico asociado al error. La propiedad del código de error se puede usar en la expresión para seleccionar sólo el error concreto que se desea controlar en esa cláusula Catch.

Ver un ejemplo de Visual Basic que ilustra la instrucción Catch/When en el sitio MSDN en [Utilizar controladores de excepciones filtrados por el usuario](http://msdn.microsoft.com/es-ar/library/4dy8x9k9(v=vs.80).aspx).

#### **Cómo:** Utilizar bloques Finally

Cuando se produce una excepción, se detiene la ejecución y da el control al controlador de excepciones más cercano. A menudo, esto significa que no se ejecutan líneas de código que se espera que se llamen siempre. Siempre se debe ejecutar cierta limpieza de recursos, como el cierre de un archivo, incluso si se produce una excepción. Para lograr esto, se puede usar un bloque Finally. Los bloques Finally se ejecutan siempre, independientemente de si se produce una excepción o no.

En el ejemplo de código siguiente se usa un bloque try/catch para detectar una ArgumentOutOfRangeException. El método Main crea dos matrices e intenta copiar una en la otra. La acción genera una ArgumentOutOfRangeException y el error se escribe en la consola. El bloque Finally se ejecuta, sea cual sea el resultado de la acción de copia.

|  |
| --- |
| // (C#) Ejemplo: crear propia especialización de clase Excepción  using System;  class ArgumentOutOfRangeExample {  static public void Main()  {  int[] array1={0,0};  int[] array2={0,0};  try  {  Array.Copy(array1,array2,-1);  }  catch (ArgumentOutOfRangeException e)  {  Console.WriteLine("Error: {0}",e);  }  finally  {  Console.WriteLine("This statement is always executed.");  }  } } |

#### Cómo: Crear clases personalizadas de Excepciones

Si desea que los usuarios puedan distinguir programaticamente, ciertas condiciones de error de otras, puede crear sus propias excepciones definidas por el usuario (user-defined).

El .NET Framework proporciona una jerarquía de clases de excepción que, en última instancia, derivan de la clase base **Exception**. Cada una de estas clases define una excepción específica, por lo que en muchos casos sólo hay que detectar la excepción que se ajuste al error que deseamos manejar, aunque también se pueden crear clases de excepción personalizadas derivadas de la clase Exception o de alguna de sus derivadas.

Cuando se creen excepciones personalizadas, es recomendable finalizar el nombre de la clase de la excepción definida por el usuario con la palabra "*Excepción*". También se recomienda implementar los tres constructores comunes recomendados, como se muestra en el ejemplo siguiente, donde se deriva una nueva clase de excepción, *EmployeeListNotFoundException*, como especialización de la clase Exception. Se definen tres constructores en la clase, cada uno con parámetros diferentes.

|  |
| --- |
| // (C#) Ejemplo: crear propia especialización de clase Excepción  using System; public class EmployeeListNotFoundException: Exception {  public EmployeeListNotFoundException()  { }  public EmployeeListNotFoundException(string message)  : base(message)  { }  public EmployeeListNotFoundException(string message, Exception inner)  : base(message, inner)  { } } |

**NOTA**: se puede ampliar con mayor este tema desde los recursos disponibles o indicados por la cátedra.

***Fuentes donde se trata el tema****:*

CEBALLOS SIERRA, Francisco. "[Microsoft C#: Lenguaje y Aplicaciones.](http://www.frro.utn.edu.ar/biblioteca_libro.php?fac=8&biblio=8&especbiblio=5&ubica=8287&palabra=C#&orden=&seccion=14&area=40)" Capítulo 6 “Clases de uso común”, pag. 106 “Excepciones”. [Nivel: básico/inicial]

### Modificadores de Acceso

Los modificadores de acceso son palabras clave que especifican la accesibilidad declarada de un miembro o un tipo. Esta sección presenta los cuatro modificadores de acceso:

* [public](http://msdn.microsoft.com/es-ar/library/yzh058ae(v=vs.80))
* [protected](http://msdn.microsoft.com/es-ar/library/bcd5672a(v=vs.80))
* [internal](http://msdn.microsoft.com/es-ar/library/7c5ka91b(v=vs.80))
* [private](http://msdn.microsoft.com/es-ar/library/st6sy9xe(v=vs.80))

Mediante los modificadores de acceso se pueden especificar los siguientes cinco niveles de accesibilidad:

**public**: acceso no restringido.

**protected**: acceso limitado a la clase contenedora o a los tipos derivados de esta clase.

**Internal**: acceso limitado al ensamblado actual.

**protected internal**: acceso limitado al ensamblado actual o los tipos derivados de la clase contenedora.

**private**: acceso limitado al tipo contenedor.

Ampliar en MSDN “[Modificadores de Acceso](http://msdn.microsoft.com/en-us/library/wxh6fsc7(v=vs.80))” y otras fuentes disponibles.

### Namespaces (Espacios de Nombres)

La palabra clave ***namespace*** se utiliza para declarar un ámbito. Este ámbito permite organizar el código y proporciona una forma de crear tipos globalmente únicos.

Dentro de un espacio de nombres, se pueden declarar uno o varios de los siguientes tipos:

* otro espacio de nombres
* [clase](http://msdn.microsoft.com/es-ar/library/0b0thckt.aspx)
* [interfaz](http://msdn.microsoft.com/es-ar/library/87d83y5b.aspx)
* [struct](http://msdn.microsoft.com/es-ar/library/ah19swz4.aspx)
* [enum](http://msdn.microsoft.com/es-ar/library/sbbt4032.aspx)
* [delegado](http://msdn.microsoft.com/es-ar/library/900fyy8e.aspx)

Los espacios de nombres disponen implícitamente de un acceso público que no puede modificarse. Para obtener una descripción de los modificadores de acceso que se pueden asignar a los elementos de un espacio de nombres, vea [Modificadores de acceso (Referencia de C#)](http://msdn.microsoft.com/es-ar/library/wxh6fsc7.aspx).

Un espacio de nombres se puede definir en dos o más declaraciones. Por ejemplo, en el siguiente ejemplo se definen dos clases como parte del espacio de nombres MyCompany:

|  |
| --- |
| // (C#) Definición de Espacios de Nombres en más de una declaración  namespace MyCompany.ProjectName {  class MyClass  { } }  namespace MyCompany.ProjectName {  class MyOtherClass  { } } |

Ampliar en MSDN “[Espacios de Nombres](http://msdn.microsoft.com/es-ar/library/z2kcy19k.aspx)” y otras fuentes disponibles.

#### Palabra reservada using

La palabra clave using tiene dos usos principales:

* Como **directiva**, cuando se utiliza para crear un alias para un espacio de nombres o para importar[[8]](#footnote-7) tipos definidos en otros espacios de nombres. Vea [Directiva using](http://msdn.microsoft.com/es-ar/library/sf0df423.aspx).
* Como **instrucción**, cuando define un ámbito al final del cual el objeto se destruye. Vea [Instrucción using](http://msdn.microsoft.com/es-ar/library/yh598w02.aspx).

Ampliar en MSDN “[using](http://msdn.microsoft.com/es-ar/library/zhdeatwt.aspx)” y otras fuentes disponibles.

### 

### Enumeraciones

### Nota IMPORTANTE: este tema no esta desarrollado aún.

Consultar entre otras fuentes en MSDN “[La clase Enum](http://msdn.microsoft.com/es-es/library/system.enum.aspx)”

### Orientación a Objetos

**Objetivo(s)**

* Introducir los elementos básicos de Programación Orientada a Objetos empleando como herramienta de desarrollo la plataforma .NET y el lenguaje C#.
* Analizar los beneficios de la herencia, del polimorfismo y el uso básico de colecciones de objetos.
* Poder modelar un problema básico utilizando Programación Orientada a Objetos, detectando clases, responsabilidades y jerarquías, y luego implementarlo utilizando C# en la plataforma .NET.

**NOTA IMPORTANTE**: **este tema NO ha sido desarrollado aun en el presente documento**. Remitirse momentáneamentee a materiales brindados por la cátedra sumados a los recursos digitales que se detallan debajo y aquellos adicionales a los que pudieran tener acceso.

*Fuente MSDN*

- **Programación orientada a objetos (C# y Visual Basic)**

Ingles: <http://msdn.microsoft.com/en-us/library/dd460654.aspx>

Traduccion automatica al espanol: <http://msdn.microsoft.com/es-ar/library/dd460654.aspx>

- **Objetos, Clases y Estructuras**

<http://msdn.microsoft.com/es-es/library/ms173109(v=vs.80)>

- **Polimorfismo**

<http://msdn.microsoft.com/es-es/library/ms173152(v=vs.80).aspx>

*Fuente Blog El Guille*

- **Programación Orientada a Objetos en .NET**

<http://www.elguille.info/NET/dotnet/POO_VB_NET_tp6.htm>

- **Programación Orientada a Objetos en .NET (2)**

<http://www.elguille.info/NET/dotnet/POO_NET_tp7.htm>

- **Conceptos y principios orientado a objetos**

<http://www.elguille.info/colabora/NET2005/Percynet_Conceptosyprincipiosorientadoaobjetos.htm>

### Ejemplos de Código

Puede ver algunos ejemplos de código de MSDN aquí: [Ejemplos de Visual C#](http://msdn.microsoft.com/es-es/library/z9hsy596(v=vs.90).aspx)

## Resumen de la Unidad / Capítulo

**IMPORTANTE:** los **contenidos** desarrollados en esta sección **no son completos**, sino que son un mini resumen de conceptos generales de los temas de mayor importancia del capítulo a modo de un repaso general de los mismos una vez completada la lectura del mismo.

Los programas deben poder controlar los **errores** que se producen durante la ejecución de manera uniforme. EL Common Language Runtime (CLR) ofrece una gran ayuda para diseñar software con tolerancia a errores mediante un modelo que informa a los programas de los errores de manera uniforme iniciando (lanzando) **excepciones**.

Una **excepción** es una alteración del flujo normal de ejecución de una aplicación.

**.Net** incorpora mecanismos para el tratamiento de las situaciones anómalas, denominadas **excepciones**, que pueden producirse durante la ejecución de un programa. Estas excepciones se controlan mediante código situado fuera del flujo normal de control mediante bloques de control que en el lenguaje C# utiliza las palabras reservadas **try**-**catch**-**finaly**.

El **sistema de manejo de excepciones** ofrecido por la plataforma tiene como **finalidad** principal permitir el **tratamiento de errores** y otras situaciones no esperadas en la ejecución de las aplicaciones.

La **BCL** contiene una jerarquía de clases de excepciones que derivan de la clase base *System.Exception* y admite ser extendida creando las propias como especialización de la clase base o algunas de sus clases más específicas derivadas de ella. Además se encuentran las clases *System.SystemException* y *System.ApplicationException*.

El orden de las instrucciones Catch es importante. Ubique los bloques Catch dirigidos a excepciones específicas antes de un bloque Catch de una excepción general, ya que serán evaluadas en forma secuencial desde la primera en adelante.

## Bibliografía[[9]](#footnote-8)

**IMPORTANTE**: Se sugiere buscar bibliografía adicional a la citada aquí, en el buscador disponible en [Biblioteca UTN Rosario](http://www.frro.utn.edu.ar/biblioteca.php).

BALENA, Francesco “[Programación avanzada con Microsoft Visual Basic.Net.](http://www.frro.utn.edu.ar/biblioteca_libro.php?fac=8&biblio=8&especbiblio=87&ubica=7471&palabra=Microsoft%20Visual%20Basic.Net&orden=&seccion=14&area=40)”, Madrid, McGraw-Hill, 2003.

ISBN: 8448137159

Capítulo 1 “Primeros pasos con visual basic.Net”, 2 “Modulos y variables”, 3 “Control de flujo y manejo de errores” y 9 “Matrices, listas y colecciones”

Ejemplares Disponibles: [en biblioteca](http://www.frro.utn.edu.ar/biblioteca_libro.php?fac=8&biblio=8&especbiblio=87&ubica=7471&palabra=visual%20basic&orden=&seccion=14&area=40)

BOOCH, Grady; RUMBAUGH, James y JACOBSON, Ivar. “[El Lenguaje Unificado de Modelado: Manual de referencia.](http://www.frro.utn.edu.ar/biblioteca_libro.php?fac=8&biblio=8&especbiblio=160&ubica=8020&palabra=object&orden=&seccion=14&area=40)” , Pearson, 2007

ISBN: 9788478290765

Ejemplares Disponibles:[en Biblioteca](http://www.frro.utn.edu.ar/biblioteca_libro.php?fac=8&biblio=8&especbiblio=160&ubica=8020&palabra=object&orden=&seccion=14&area=40)

CEBALLOS SIERRA, Francisco. "[El lenguaje de](http://www.frro.utn.edu.ar/biblioteca_libro.php?fac=8&biblio=8&especbiblio=87&ubica=7781&palabra=.net&orden=&seccion=14&area=40) [programación](http://www.google.com/url?q=http%3A%2F%2Fwww.frro.utn.edu.ar%2Fbiblioteca_libro.php%3Ffac%3D8%26biblio%3D8%26especbiblio%3D87%26ubica%3D7781%26palabra%3D.net%26orden%3D%26seccion%3D14%26area%3D40&sa=D&sntz=1&usg=AFQjCNF-AJcc_98tZLJFn1zdKWXpw10I4g)[n Visual Basic.Net.](http://www.frro.utn.edu.ar/biblioteca_libro.php?fac=8&biblio=8&especbiblio=87&ubica=7781&palabra=.net&orden=&seccion=14&area=40)", Mexico, Alfaomega, 2002.

ISBN: 9701508297

Capítulos:

2. Fundamentos de Visual Basic.NET .

3. Programación orientada a objetos.

4. Elementos del lenguaje.

5. Estructura de un programa.

6. Clases de uso comun.

7. Sentencias de control.

8. Matrices.

9. Más sobre procedimientos.

Ejemplares Disponibles:[en Biblioteca](http://www.frro.utn.edu.ar/biblioteca_libro.php?fac=8&biblio=8&especbiblio=87&ubica=7781&palabra=.net&orden=&seccion=14&area=40)

CEBALLOS SIERRA, Francisco. "[Microsoft C#: Curso de programacion.](http://www.frro.utn.edu.ar/biblioteca_libro.php?fac=8&biblio=8&especbiblio=5&ubica=8288&palabra=C#&orden=&seccion=14&area=40)", México, Alfaomega, 2007.

ISBN: 9701512227

Capítulos:

2. Fundamentos de C#.

3. Programación orientada a objetos.

4. Elementos del lenguaje.

5. Estructura de un programa.

6. Clases de uso común.

7. Sentencias de control.

8. Matrices.

Ejemplares Disponibles:[en Biblioteca](http://www.frro.utn.edu.ar/biblioteca_libro.php?fac=8&biblio=8&especbiblio=5&ubica=8288&palabra=C#&orden=&seccion=14&area=40)

CEBALLOS SIERRA, Francisco. "[Microsoft C#: Lenguaje y Aplicaciones.](http://www.frro.utn.edu.ar/biblioteca_libro.php?fac=8&biblio=8&especbiblio=5&ubica=8287&palabra=C#&orden=&seccion=14&area=40)", México, Alfaomega, 2008.

ISBN: 9789701513712

Capítulos:

2. Fundamentos e C#.

3. Programación orientada a objetos.

4. Elementos del lenguaje.

5. Estructura de un programa.

6. Clases de uso común.

7. Sentencias de control.

8. Matrices.

9. Más sobre métodos.

Ejemplares Disponibles:[en Biblioteca](http://www.frro.utn.edu.ar/biblioteca_libro.php?fac=8&biblio=8&especbiblio=5&ubica=8287&palabra=C#&orden=&seccion=14&area=40)

DEITEL, Harvey y DEITEL, Paul. "[Como programar en C#.](http://www.frro.utn.edu.ar/biblioteca_libro.php?fac=8&biblio=8&especbiblio=5&ubica=8320&palabra=C#&orden=&seccion=14&area=40)", Mexico, Pearson Educacion, 2007.

ISBN: 9789702610564

Capítulo 3 "Introducción a las aplicaciones de C#", 4 "Introducción a las clases y los objetos", 5 "Instrucciones de control: parte 1", 6 "Instrucciones de control: parte 2", 7 "Métodos: un análisis más detallado", 8 "Arreglos", 9 "Clases y objetos: un análisis más detallado", 10 "Programación orientada a objetos: herencia", 11 "Polimorfismo, interfaces y sobrecarga de operadores", 12 "Manejo de excepciones", 16 "Cadenas, caracteres y expresiones regulares", 24 "Estructuras de datos.", 25 "Genéricos", 26 "Colecciones" y Apéndice A "Tabla de precedencia de los operadores"

Ejemplares Disponibles:[en Biblioteca](http://www.frro.utn.edu.ar/biblioteca_libro.php?fac=8&biblio=8&especbiblio=5&ubica=8320&palabra=C#&orden=&seccion=14&area=40)

FOXALL, James “[Visual C# 2005.](http://www.frro.utn.edu.ar/biblioteca_libro.php?fac=8&biblio=8&especbiblio=5&ubica=8553&palabra=C#&orden=&seccion=14&area=40)”, Madrid, Anaya, 2006.

ISBN: 9788441521216

Capítulos

2. Explorar visual C# 2005.

3. Objetos y colecciones.

10. Métodos: Creación y llamadas.

11. Utilizar constantes, tipos de datos, variables y matrices.

12. Cálculos aritméticos, manipulación de cadenas y ajustes de fecha y hora.

13. Tomas decisiones en visual C#.

14. Bucles para obtener eficiencia.

15. Depuración del código.

16. Diseñar objetos con clases.

Disponibilidad: [en biblioteca](http://www.frro.utn.edu.ar/biblioteca_libro.php?fac=8&biblio=8&especbiblio=5&ubica=8553&palabra=C#&orden=&seccion=14&area=40)

FIRESMITH, Donald. “[Object-oriented requeriments analysis and design: A software approach.](http://www.frro.utn.edu.ar/biblioteca_libro.php?fac=8&biblio=8&especbiblio=186&ubica=7574&palabra=object&orden=&seccion=14&area=40)”, New York, Wiley & Sons, 1992.

ISBN: 047157807x

Ejemplares Disponibles:[en Biblioteca](http://www.frro.utn.edu.ar/biblioteca_libro.php?fac=8&biblio=8&especbiblio=186&ubica=7574&palabra=object&orden=&seccion=14&area=40)

HILYARD, Jayy TEILHET, Stephen "[C# 3.0:](http://www.frro.utn.edu.ar/biblioteca_libro.php?fac=8&biblio=8&especbiblio=5&ubica=28614&palabra=C#&orden=&seccion=14&area=40) [Guí](http://www.google.com/url?q=http%3A%2F%2Fwww.frro.utn.edu.ar%2Fbiblioteca_libro.php%3Ffac%3D8%26biblio%3D8%26especbiblio%3D5%26ubica%3D28614%26palabra%3DC%23%26orden%3D%26seccion%3D14%26area%3D40&sa=D&sntz=1&usg=AFQjCNH8Jz7OkJ9ys1avCcHV9LYB-ZaCcg)[a de referencia.](http://www.frro.utn.edu.ar/biblioteca_libro.php?fac=8&biblio=8&especbiblio=5&ubica=28614&palabra=C#&orden=&seccion=14&area=40)", Madrid, Anaya Multimedia, 2009.

ISBN: 9788441524910

Capítulo 2 "Cadenas y caracteres.", 3 "Clases y estructuras.", 4 "Genéricos.", 5 "Colecciones.", 7 "Gestión de excepciones.", 10 "Expresiones regulares.", 11 "Estructuras de datos y algoritmos." y 20 "Números y enumeraciones."

Ejemplares Disponibles:[en Biblioteca](http://www.frro.utn.edu.ar/biblioteca_libro.php?fac=8&biblio=8&especbiblio=5&ubica=28614&palabra=C#&orden=&seccion=14&area=40)

RAMIREZ, Felipe. "[Programación](http://www.google.com/url?q=http%3A%2F%2Fwww.frro.utn.edu.ar%2Fbiblioteca_libro.php%3Ffac%3D8%26biblio%3D8%26especbiblio%3D5%26ubica%3D8506%26palabra%3DC%23%26orden%3D%26seccion%3D14%26area%3D40&sa=D&sntz=1&usg=AFQjCNEZcVk44yzF7JFuaGsgZT8NhlXHJg)[: algoritmos y su](http://www.frro.utn.edu.ar/biblioteca_libro.php?fac=8&biblio=8&especbiblio=5&ubica=8506&palabra=C#&orden=&seccion=14&area=40) [implementació](http://www.google.com/url?q=http%3A%2F%2Fwww.frro.utn.edu.ar%2Fbiblioteca_libro.php%3Ffac%3D8%26biblio%3D8%26especbiblio%3D5%26ubica%3D8506%26palabra%3DC%23%26orden%3D%26seccion%3D14%26area%3D40&sa=D&sntz=1&usg=AFQjCNEZcVk44yzF7JFuaGsgZT8NhlXHJg)[n en Vb.Net, C#, Java y C++.](http://www.frro.utn.edu.ar/biblioteca_libro.php?fac=8&biblio=8&especbiblio=5&ubica=8506&palabra=C#&orden=&seccion=14&area=40)", México, Alfaomega, 2007.

ISBN: 9789701512807

Capítulos:

10. Implementación de algoritmos en Visual Basic.Net.

11. Implementación de algoritmos en C#.

14. Fundamentos de la programación orientada a objetos.

Ejemplares Disponibles:[en Biblioteca](http://www.frro.utn.edu.ar/biblioteca_libro.php?fac=8&biblio=8&especbiblio=5&ubica=8506&palabra=C#&orden=&seccion=14&area=40)

SHARP, John. “[Visual C# 2008: paso a paso.](http://www.frro.utn.edu.ar/biblioteca_libro.php?fac=8&biblio=8&especbiblio=5&ubica=8551&palabra=Visual%20C#%202008&orden=&seccion=14&area=40)” Madrid, Anaya, 2008.

ISBN: 9788441524491

Capítulos:

1 “Bienvenido a C#”

Disponibilidad: [en biblioteca](http://www.frro.utn.edu.ar/biblioteca_libro.php?fac=8&biblio=8&especbiblio=5&ubica=8551&palabra=Visual%20C#%202008&orden=&seccion=14&area=40)

SPENCER, Ken ; EBARDHARD, Tom y ALEXANDER, John. "[OOP : Buildind reusable componentes with Microsoft Visual Basic.Net](http://www.frro.utn.edu.ar/biblioteca_libro.php?fac=8&biblio=8&especbiblio=87&ubica=7997&palabra=.net&orden=&seccion=14&area=40)", Washington, Microsoft Press, 2003.

ISBN: 0735613796

Ejemplares Disponibles:[en Biblioteca](http://www.frro.utn.edu.ar/biblioteca_libro.php?fac=8&biblio=8&especbiblio=87&ubica=7997&palabra=.net&orden=&seccion=14&area=40)

**Fuentes Digitales**



+ Microsoft Developer Network (MSDN) On Line:<http://msdn.microsoft.com>

Referencia C#: <http://msdn.microsoft.com/en-us/library/618ayhy6(v=vs.80).aspx>

Manejo y Lanzamiento de Excepciones:

<http://msdn.microsoft.com/en-us/library/5b2yeyab(v=vs.80).aspx>

Clase Exception: <http://msdn.microsoft.com/en-us/library/system.exception.aspx>

+ Bloge de El Guille: [http://www.elguille.info](http://www.elguille.info/)

+ Wikipedia (conceptos genéricos y específicos)

Programación Orientada a Objetos:

<http://en.wikipedia.org/wiki/Object-oriented_programming>

<http://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos>

Lengujes Orientados a Objetos:

### Anexo A: Equivalencias y diferencias entre lenguajes .Net

Se detallan una serie de contenidos y recursos donde se plantean las diferencias principales existentes entre los lenguajes más difundidos de la plataforma .Net.

**Language Equivalentes**

(Aplicado a VS 2008 y otras versiones anteriores y posteriores disponibles)

<http://msdn.microsoft.com/en-us/library/czz35az4(vs.71).aspx>

**Equivalencias entre Visual Basic .NET y C#**

Presenta una guía general de referencia sobre equivalencias entre los dos lenguajes más utilizados, C# y VB.Net

(Última Actualización: Agosto 2006)

<http://www.elguille.info/NET/dotnet/equivalenciavbcs1.htm>

**Differences Between Visual Basic .NET and Visual C# .NET** (white paper)

(Última Actualización: Abril 2007)

<http://support.microsoft.com/kb/308470>

**Comparison of C Sharp and Visual Basic .NET**

(Última Actualización: Abril 2012)

<http://en.wikipedia.org/wiki/Comparison_of_C_Sharp_and_Visual_Basic_.NET>

**Comparison of C Sharp and Java**

<http://en.wikipedia.org/wiki/Comparison_of_C_Sharp_and_Java>

**Historial de Versiones**

|  |  |  |  |
| --- | --- | --- | --- |
| **Versión** | **Fecha** | **Autor** | **Detalle** |
| 1.0 | 15/03/2012 | Ezequiel Porta | Versión Inicial de contenidos de la Unidad 3 “Sintaxis y Orientación a Objetos de Lenguajes .Net” sobre tema Manejo de Errores y Excepciones (Try/Catch/Finally). |
| 1.1 | 10/04/2012 | Ezequiel Porta | Anexo de contenidos y referencias sobre equivalencias y diferencias entre lenguajes .Net  Lista de enlaces y referencia a documentación de cátedra sobre el tema Orientación a Objetos. |
|  |  |  |  |
|  |  |  |  |

1. Puede ver un detalle de los “[Lenguajes de Progrmacion](http://msdn.microsoft.com/es-es/library/aa292164(v=vs.71).aspx)” .Net en el sitio del MSDN [↑](#footnote-ref-0)
2. Lenguaje totalmente nuevo con la aparición de .Net cuya sintaxis tiene muchas similitudes a C con una plena implementación del Paradigmas Orientado a Objetos (POO). [↑](#footnote-ref-1)
3. [Lenguaje funcional](http://msdn.microsoft.com/es-es/magazine/cc164244.aspx) fue creado por Don Syme del equipo de investigación de Microsoft. [↑](#footnote-ref-2)
4. Ofrecer un diseño de software con tolerancia a errores. [↑](#footnote-ref-3)
5. Componente del CLR conocido por su termino en ingles como *Exception Manager* o *Exception Handling*. [↑](#footnote-ref-4)
6. Vea en MSDN “[Como administra las excepciones el motor de ejecución](http://msdn.microsoft.com/es-ar/library/6kzk0czb(v=vs.80).aspx)” [↑](#footnote-ref-5)
7. Puede ampliar su conocimiento en MSDN sobre la clase [System.Exception](http://msdn.microsoft.com/en-us/library/system.exception.aspx). Además puede consultar estas otras clases del namespace System: [SystemException](http://msdn.microsoft.com/es-ar/library/system.systemexception.aspx) y [ApplicationException](http://msdn.microsoft.com/en-us/library/system.applicationexception.aspx). [↑](#footnote-ref-6)
8. En Visual Basic .Net la palabra reservada equivalente a ***using*** es justamente ***Imports***. [↑](#footnote-ref-7)
9. La Bibliografía está organizada en una primer sección de libros impresos, ordenados alfabéticamente por apellido del autor, muchos de ellos disponibles en Biblioteca de UTN Rosario en cuyo caso aparecerá un enlace a los datos de la publicación que figuran en línea en el buscador del sitio web de la biblioteca. Seguido se detallan aquellos recursos digitales ya sean otros libros disponibles en dicho formato que podrá encontrar en el aula virtual o de alguna otra fuente de información como pueden ser los libros en línea del MSDN del sitio oficial de Microsoft o sitios de empresas o profesionales que poseen contenidos de valor. [↑](#footnote-ref-8)