



© JMA 2016. All rights reserved

## Enlaces

---

- Buenas prácticas:
  - <https://docs.microsoft.com/es-es/dotnet/standard/design-guidelines/>
  - <https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/inside-a-program/coding-conventions>

---

© JMA 2016. All rights reserved

# INTRODUCCIÓN A .NET FRAMEWORK

© JMA 2016. All rights reserved

## Retos del desarrollo en Microsoft

- Integración de aplicaciones
  - Múltiples lenguajes de programación
  - Múltiples modelos de programación
  - Complejidad del desarrollo y despliegue
  - Seguridad no inherente
- 
- Preservar la inversión del desarrollador
  - Elevar la productividad del desarrollador

© JMA 2016. All rights reserved

## ¿Qué es .NET?

- Plataforma de Desarrollo compuesta de
  - Entorno de Ejecución (Runtime)
  - Bibliotecas de Funcionalidad (Class Library)
  - Lenguajes de Programación
  - Compiladores
  - Herramientas de Desarrollo (IDE & Tools)
  - Guías de Arquitectura
- La evolución de la plataforma COM

© JMA 2016. All rights reserved

## Características de .NET

- Plataforma de ejecución intermedia
- 100% Orientada a Objetos
- Multilenguaje
- Plataforma Empresarial de Misión Crítica
- Modelo de Programación único para todo tipo de aplicaciones y dispositivos de hardware
- Se integra fácilmente con aplicaciones existentes desarrolladas en plataformas Microsoft
- Se integra fácilmente con aplicaciones desarrolladas en otras plataformas

© JMA 2016. All rights reserved

## ¿Qué es el .NET?

- Paquete de software fundamental de la plataforma .NET. Incluye:
  - Entorno de Ejecución (Runtime)
  - Bibliotecas de Funcionalidad (Class Library)
- Se distribuye en forma libre y gratuita
- Existen tres variantes principales:
  - .NET Framework Redistributable Package
  - .NET Framework SDK
  - .NET Compact Framework
- Está instalado por defecto en Windows 2003 Server o superior

© JMA 2016. All rights reserved

## Implementaciones de .NET

- Una aplicación de .NET se desarrolla y se ejecuta en una o varias implementaciones de .NET. Las implementaciones de .NET incluyen .NET Framework, .NET Core y Mono. Hay una especificación de API común a todas las implementaciones de .NET que se denomina .NET Standard.
- Cada implementación de .NET incluye los siguientes componentes:
  - Uno o varios entornos de ejecución. Ejemplos: CLR para .NET Framework, CoreCLR y CoreRT para .NET Core.
  - Una biblioteca de clases que implementa .NET Standard y puede implementar API adicionales. Ejemplos: biblioteca de clases base de .NET Framework, biblioteca de clases base de .NET Core.
  - Opcionalmente, uno o varios marcos de trabajo de la aplicación. Ejemplos: ASP.NET, Windows Forms y Windows Presentation Foundation (WPF) se incluyen en .NET Framework y .NET Core.
  - Opcionalmente, herramientas de desarrollo. Algunas herramientas de desarrollo se comparten entre varias implementaciones.
- Hay cinco implementaciones principales de .NET que Microsoft desarrolla y mantiene activamente:
  - .NET Framework, .NET Core, Mono, UWP y MAUI.

© JMA 2016. All rights reserved

# .NET Framework

- También conocida como .NET "Full Framework" o .NET "Tradicional".
- Se trata de la plataforma .NET "de toda la vida", aparecida oficialmente en 2001. Monolítica (instalas todo su núcleo o no instalas nada), pero tremendamente capaz, ya que con ella puedes crear aplicaciones de cualquier tipo: de consola, de escritorio, para la web, móviles... Casi cualquier cosa que puedas imaginar.
- Eso sí, se trata de aplicaciones que se ejecutarán solamente sobre Windows y está optimizado para crear aplicaciones de escritorio de Windows.
- Las versiones 4.5 y posteriores implementan .NET Standard, de forma que el código que tiene como destino .NET Standard se puede ejecutar en esas versiones de .NET Framework y existen miles de componentes de terceros para poder extenderla. La versión 4.8 es la última que saldrá.
- Con ella puedes utilizar también infinidad de lenguajes de programación: C#, Visual Basic .NET, F#, C++, Python... ¡Hasta Cobol!
- La utilizan miles de empresas en todo el mundo. No va a cambiar mucho en los próximos años.

© JMA 2016. All rights reserved

# .NET Core

- .NET Core 1.0 fue anunciado el 12 de noviembre de 2014 y se lanzó el 27 de junio de 2016. Es una nueva plataforma, escrita desde cero con varios objetivos en mente, siendo los principales:
  - Más ligera y "componentizable": de modo que con nuestra aplicación se distribuya exclusivamente lo que necesitemos, no la plataforma completa. En cuanto a los lenguajes disponibles, podremos utilizar C#, F# y Visual Basic .NET.
  - Multi-plataforma: las aplicaciones que creamos funcionarán en Windows, Linux y Mac, no solo en el sistema de Microsoft.
  - Alto rendimiento: no es que .NET tradicional no tuviese buen rendimiento, pero es que .NET Core está específicamente diseñada para ello. .NET Core tiene un desempeño más alto que la versión tradicional (minimiza dependencias y servicios adicionales) lo cual es muy importante para entornos Cloud, en donde esto se traduce en mucho dinero al cabo del tiempo.
  - Pensada para la nube: cuando .NET se diseñó a finales de los años 90 el concepto de nube ni siquiera existía. .NET Core nace en la era Cloud, por lo que está pensada desde el principio para encajar en entornos de plataforma como servicio y crear aplicaciones eficientes para su funcionamiento en la nube (sea de Microsoft o no).

© JMA 2016. All rights reserved

## .NET Standard

- .NET Standard es un conjunto de API que se implementan mediante la biblioteca de clases base de una implementación de .NET, que constituyen un conjunto uniforme de contratos contra los que se compila el código.
- Estos contratos se implementan en cada implementación de .NET. Esto permite la portabilidad entre diferentes implementaciones de .NET, de forma que el código se puede ejecutar en cualquier parte.
- .NET Standard es también una plataforma de destino. Si el código tiene como destino una versión de .NET Standard, se puede ejecutar en cualquier implementación de .NET que sea compatible con esa versión de .NET Standard.

© JMA 2016. All rights reserved

## Mono (Xamarin)

- Mono es una implementación de .NET (2004) que se usa principalmente cuando se requiere un entorno de ejecución pequeño. Es el entorno de ejecución que activa las aplicaciones Xamarin en Android, macOS, iOS, tvOS y watchOS, y se centra principalmente en una superficie pequeña. Mono también proporciona juegos creados con el motor de Unity.
- Admite todas las versiones de .NET Standard publicadas actualmente.
- Históricamente, Mono implementaba la API de .NET Framework más grande y emulaba algunas de las funciones más populares en Unix. A veces, se usa para ejecutar aplicaciones de .NET que se basan en estas capacidades en Unix.
- Mono se suele usar con un compilador Just-In-Time, pero también incluye un compilador estático completo (compilación Ahead Of Time) que se usa en plataformas como iOS.

© JMA 2016. All rights reserved

## Plataforma universal de Windows (UWP)

- UWP es una implementación de .NET que se usa para compilar aplicaciones Windows modernas y táctiles, así como software para Internet de las cosas (IoT).
- Se ha diseñado para unificar los diferentes tipos de dispositivos de destino (Windows 10), incluidos equipos, tabletas, teléfonos e incluso la consola Xbox.
- UWP proporciona muchos servicios, como una tienda de aplicaciones centralizada, un entorno de ejecución (AppContainer) y un conjunto de API de Windows para usar en lugar de Win32 (WinRT).
- Pueden escribirse aplicaciones en C++, C#, Visual Basic y JavaScript. Al usar C# y Visual Basic, .NET Core proporciona las API de .NET.

© JMA 2016. All rights reserved

## Multi-platform App UI (.NET MAUI)

- .NET Multi-platform App UI (.NET MAUI) es un marco multiplataforma para crear aplicaciones móviles y de escritorio nativas con C# y XAML.
- Con .NET MAUI, se pueden desarrollar aplicaciones que se pueden ejecutar en Android, iOS, macOS y Windows desde una sola base de código compartida.
- .NET MAUI es de código abierto y es la evolución de Xamarin.Forms, extendida desde escenarios móviles a escritorio, con controles de interfaz de usuario
- Con .NET MAUI, puede crear aplicaciones multiplataforma mediante un único proyecto, pero puede agregar recursos y código fuente específicos de la plataforma si es necesario.
- Uno de los objetivos clave de .NET MAUI es permitirle implementar la mayor parte de la lógica de la aplicación y el diseño de la interfaz de usuario lo más posible en una sola base de código.

© JMA 2016. All rights reserved

## .NET Core o .NET Framework

- Usar .NET Core para la aplicación de servidor cuando:
  - Tenga necesidades multiplataforma.
  - Tenga los microservicios como objetivo.
  - Vaya a usar contenedores de Docker.
  - Necesite sistemas escalables y de alto rendimiento.
  - Necesite versiones de .NET en paralelo por aplicación.
- Usar .NET Framework para su aplicación de servidor cuando:
  - La aplicación usa actualmente .NET Framework (la recomendación es extender en lugar de migrar).
  - La aplicación usa bibliotecas .NET de terceros o paquetes de NuGet que no están disponibles para .NET Core.
  - La aplicación usa tecnologías de .NET que no están disponibles para .NET Core.
  - La aplicación usa una plataforma que no es compatible con .NET Core. Windows, macOS y Linux admiten .NET Core.

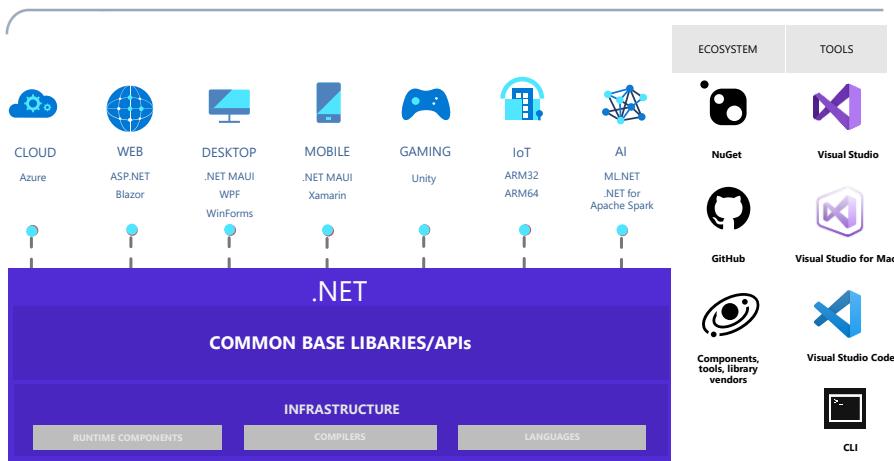
© JMA 2016. All rights reserved

## .NET

- En 2019 se detiene la evolución de .NET Framework siendo la 4.8 la última en salir y ya solo recibirá actualizaciones de seguridad. .NET Standard 2.1 es la última versión que saldrá.
- La última versión con el nombre de .NET Core es la .NET Core 3.1, a partir de la cual se denomina .NET, siendo .NET 5 la primera versión con el nuevo nombre.
- .NET 5 y versiones posteriores adoptan un enfoque diferente para establecer la uniformidad que elimina la necesidad de .NET Standard en la mayoría de los escenarios aunque seguirán admitiendo .NET Standard 2.1.
- .NET 5 y versiones posteriores están unificando todas las plataformas, incorporando MAUI y los diferentes marco (ASP.NET Core, EF Core, ...) que se han unificado en .NET 8.

© JMA 2016. All rights reserved

# .NET



© JMA 2016. All rights reserved

## Multiplataforma

- Se pueden crear aplicaciones .NET para muchos sistemas operativos, entre los que se incluyen los siguientes:
  - Windows
  - macOS
  - Linux
  - Android
  - iOS
  - tvOS
  - watchOS
- Las arquitecturas de procesador compatibles incluyen las siguientes:
  - x64
  - x86
  - ARM32
  - ARM64

© JMA 2016. All rights reserved

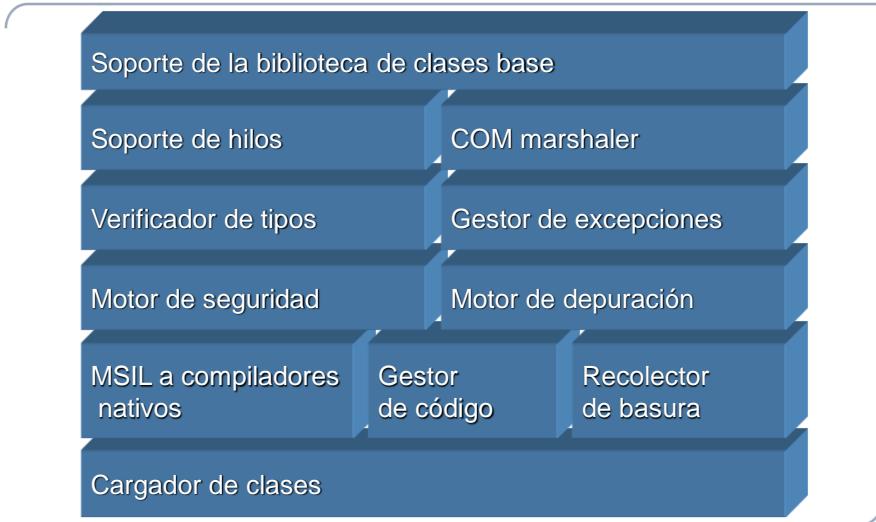
## .NET Framework en contexto



## Common Language Runtime

- El CLR es el entorno donde se ejecutan todas las aplicaciones .NET
- El CLR determina para las aplicaciones .NET:
  - Un conjunto de tipos de datos: CTS
  - Un lenguaje intermedio: CIL (MSIL)
  - Un empaquetado de código: Assembly
- El código que ejecuta el CLR se llama código administrado (managed code)

# Common Language Runtime



© JMA 2016. All rights reserved.

## Servicios suministrados por el CLR

- Compilador MSIL a nativo: Transforma código intermedio de alto nivel independiente del hardware que lo ejecuta a código de máquina propio del dispositivo que lo ejecuta.
- Cargador de Clases: Permite cargar en memoria las clases.
- Recolector de Basura: Elimina de memoria objetos no utilizados.
- Motor de Seguridad: Administra la seguridad del código que se ejecuta.
- Motor de Depuración: Permite hacer un seguimiento de la ejecución del código aún cuando se utilicen lenguajes distintos.
- Verificador de Tipos: Controla que las variables de la aplicación usen el área de memoria que tienen asignado.
- Empaquetador de COM: Coordina la comunicación con los componentes COM para que puedan ser usados por el .NET Framework.
- Administrador de Excepciones: Maneja los errores que se producen durante la ejecución del código.
- Soporte de multiproceso (threads): Permite ejecutar código en forma paralela.
- Soporte de la Biblioteca de Clases Base: Interfase con las clases base del .NET Framework.
- Administrador de Código: Coordina toda la operación de los distintos subsistemas del Common Language Runtime.

© JMA 2016. All rights reserved.

## Common Language Runtime Beneficios

- Entorno de ejecución robusto
- Seguridad inherente
- Desarrollo simplificado
- Fácil gestión y despliegue de aplicaciones
- Preserva inversión de desarrollador
- Desarrollos multiplataforma

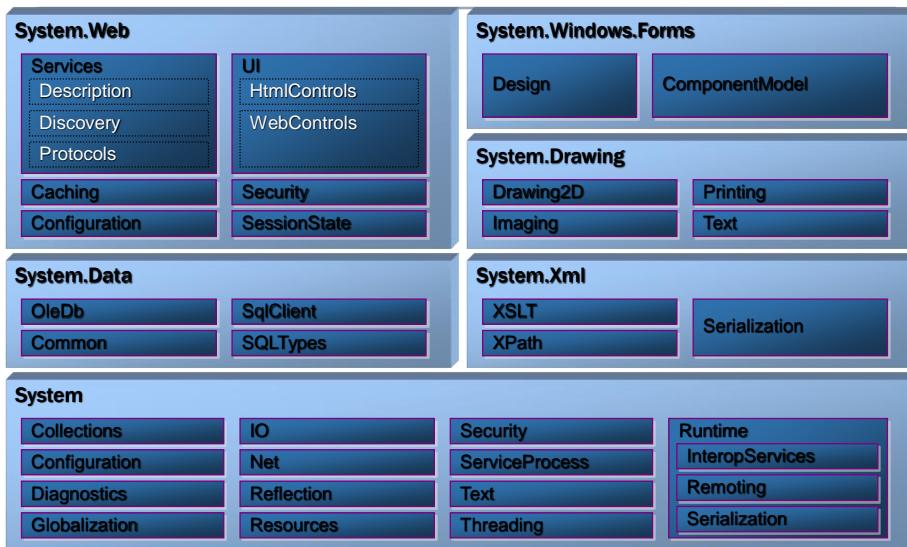
© JMA 2016. All rights reserved

## .NET Framework Class Library

- Conjunto de Tipos básicos (clases, interfaces, etc.) que vienen incluidos en el .NET Framework
- Los tipos están organizados en jerarquías lógicas de nombres, denominados NAMESPACES (Espacios de nombres)
- Los tipos son INDEPENDIENTES del lenguaje de desarrollo
- Es extensible y totalmente orientada a objetos

© JMA 2016. All rights reserved

# Librería de clases



## .NET Framework Class Library Base Class Library

- Implementadas en el propio CLR
  - Hilos, sincronización
  - Application Domains
  - ...
- Implementadas en código administrado
  - Ficheros
  - Red
  - Criptografía
  - ...

## .NET Framework Class Library

### Beneficios

- Completa, Organizada, Extensible
- Para cualquier Arquitectura de Aplicación
  - Acceso a Datos
    - ADO.NET
    - XML
  - Lógica de Negocio
    - Enterprise Services (COM+)
    - Servicios Web XML
    - .NET Remoting
  - Presentación
    - Windows Forms, WPF, UWP, XBOX,
    - Smart Client, Xamarin
    - Web Forms y Mobile Web Forms

© JMA 2016. All rights reserved

## Desarrollo de Software

- Pasos del desarrollo y ejecución administrada
  1. Elegir un lenguaje de programación
  2. Elegir un compilador (Common Language Specification)
  3. Seleccionar el pool de tecnologías
  4. Desarrollar la aplicación
  5. Compilar el código al Lenguaje Intermedio (MSIL)
  6. Compilar MSIL a código nativo
  7. Ejecutar el código administrado

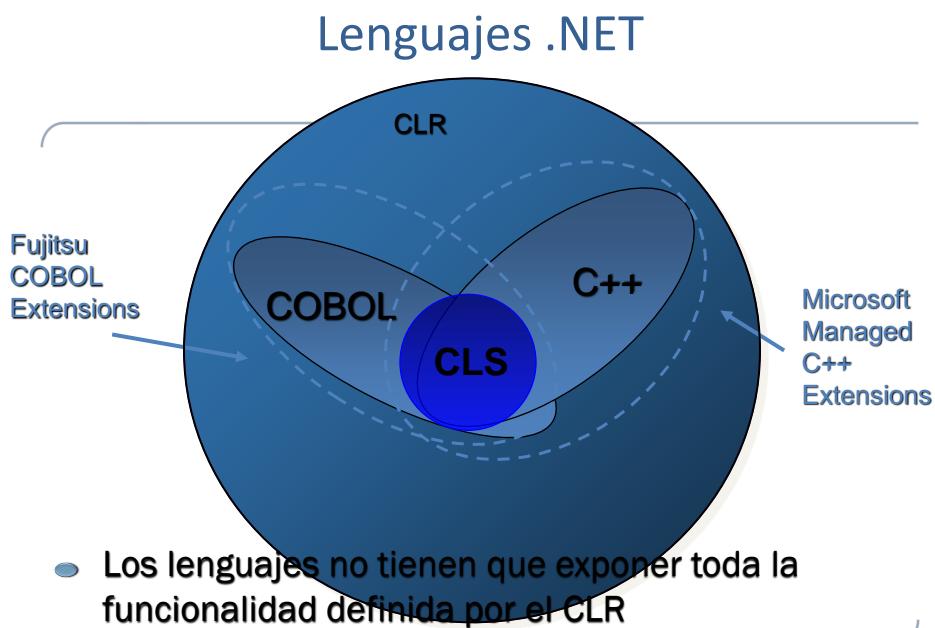
© JMA 2016. All rights reserved

## Lenguajes .NET

### CLS (Common Language Specification)

- Especificación que estandariza una serie de características soportadas por el CLR
- Requisitos mínimos para compiladores de lenguajes .NET
  - Conjunto mínimo de funcionalidad que deben implementar
- Un conjunto de tipos de datos:
  - CTS: Sistema de tipos común
- Su objetivo es facilitar la interoperabilidad entre lenguajes

© JMA 2016. All rights reserved



© JMA 2016. All rights reserved

## Múltiples lenguajes soportados

- .NET es neutral con respecto al lenguaje
- Microsoft suministra:
  - Visual C# .NET, Visual Basic .NET, Visual C++ .NET,
  - JScript, Visual J# .NET, Visual # .NET, F#
- Terceros suministran:
  - COBOL, RPG, APL, Perl, Pascal, Smalltalk, Eiffel, Fortran, Haskell, Mercury, Oberon, Oz, Python, Scheme, Standard ML, ... hasta +26 lenguajes

© JMA 2016. All rights reserved

## Lenguajes .NET: Comparativa

Lenguaje	Código gestionado	Código type-safe	Llamadas a código no gestionado	Código no gestionado
VB.NET	Sí	Siempre	Sí	No
C#	Sí	Opcional	Sí	No
C++	Sí	Nunca	Sí	Sí
J#	Sí	Siempre	Sí	No

### Otros

APL, Cobol, Component Pascal, Delta Forth, compiler, Eiffel, Fortran, Haskell, Mercury, Oberon, PERL, Python, Salford FTN95, Scheme SmallScript, Standard ML ,TMT Pascal, F#, AVR, ASML

© JMA 2016. All rights reserved

## CLR - MSIL

```
.method private hidebysig static void Main(string[] args) cil
    managed {
.entrypoint
maxstack 8
L_0000: ldstr "Hola Mundo"
L_0005: call void
    [mscorlib]System.Console.WriteLine(string)
L_000a: ret
}
```

© JMA 2016. All rights reserved.

## Soporte multilenguaje

**VB.NET**

```
Dim s as String
s = "authors"
Dim cmd As New SqlCommand("select * from " & s,
                           sqlconn)
cmd.ExecuteReader()
```

**C#**

```
string s = "authors";
SqlCommand cmd = new SqlCommand("select * from "+s,
                               sqlconn);
cmd.ExecuteReader();
```

**C++**

```
String *s = S"authors";
SqlCommand cmd = new
SqlCommand(S"select * from ", s),
           sqlconn);
cmd.ExecuteReader();
```

© JMA 2016. All rights reserved.

## Soporte multilenguaje

```
String s = "authors";
SqlCommand cmd = new SqlCommand("select * from "+s,
                                sqlconn);
cmd.ExecuteReader();
```

**J#**

```
var s = "authors"
var cmd = new SqlCommand("select * from " + s, sqlconn)
cmd.ExecuteReader()
```

**JScript**

```
String *s = S"authors";
SqlCommand cmd = new SqlCommand(String::Concat
                                (S"select * from ", s), sqlconn);
cmd.ExecuteReader();
```

**Perl**

© JMA 2016. All rights reserved.

## Soporte multilenguaje

**Cobol**

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
  CLASS SqlCommand AS "System.Data.SqlClient.SqlCommand"
  CLASS SqlConnection AS "System.Data.SqlClient.SqlConnection".
DATA DIVISION.
WORKING-STORAGE SECTION.
01 str PIC X(50).
01 cmd-string PIC X(50).
01 cmd OBJECT REFERENCE SqlCommand.
01 sqlconn OBJECT REFERENCE SqlConnection.
PROCEDURE DIVISION.
  *> Establish the SQL connection here somewhere.
MOVE "authors" TO str.
STRING "select * from " DELIMITED BY SIZE,
      str DELIMITED BY " " INTO cmd-string.
INVOKE SqlCommand "NEW" USING BY VALUE cmd-string sqlconn RETURNING cmd.
INVOK cmd "ExecuteReader".
```

© JMA 2016. All rights reserved.

## Soporte multilenguaje

### RPG

```
DclFld MyInstObj Type( System.Data.SqlClient.SqlCommand )
DclFld s Type( *string )
s = "authors"
MyInstObj = New System.Data.SqlClient.SqlCommand("select *
      from "+s, sqlconn)
MyInstObj.ExecuteReader()
```

### Fortran

```
assembly_external(name="System.Data.SqlClient.SqlCommand")
sqlcmdcharacter*10 xsqcmd
Cmd x='authors'
cmd = sqcmd("select * from "//x, sqlconn)
call cmd.ExecuteReader()
end
```

© JMA 2016. All rights reserved.

## Lenguajes .NET

### Beneficios

- Independencia de las aplicaciones del lenguaje de programación utilizado
- Desarrollo de aplicaciones multi-lenguaje
- Preserva inversión del desarrollador
- Facilita adopción de .NET
- El lenguaje determina la productividad, no la eficiencia

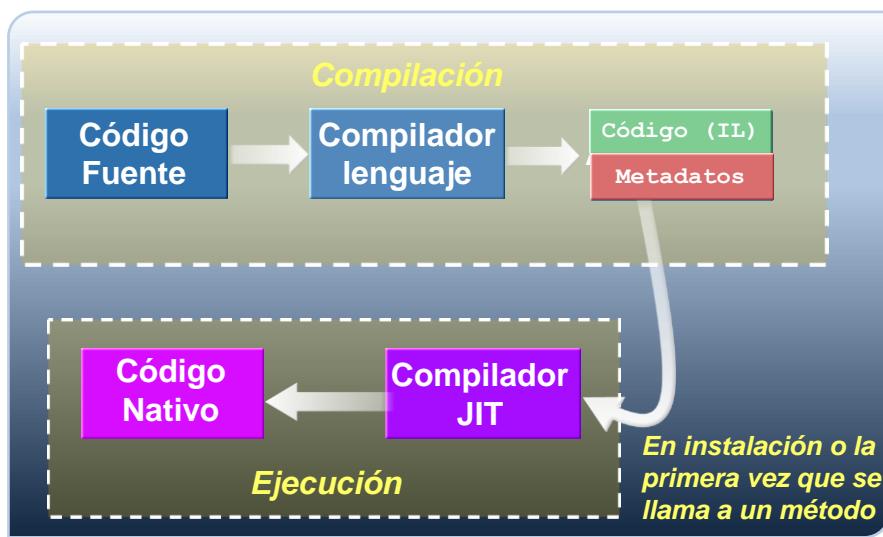
© JMA 2016. All rights reserved.

## CLS - Elección del lenguaje

- .NET posee un único runtime (el CLR) y un único conjunto de bibliotecas para todos los lenguajes
- No hay diferencias notorias de rendimientos entre los lenguajes provistos por Microsoft
- El lenguaje a utilizar, en general, dependerá de la experiencia previa con otros lenguajes o de gustos personales
  - Si conoce Java, Delphi, C++, etc. → C#
  - Si conoce Visual Basic o VBScript → VB.NET
- Los tipos de aplicaciones .NET son INDEPENDIENTES del lenguaje que elija

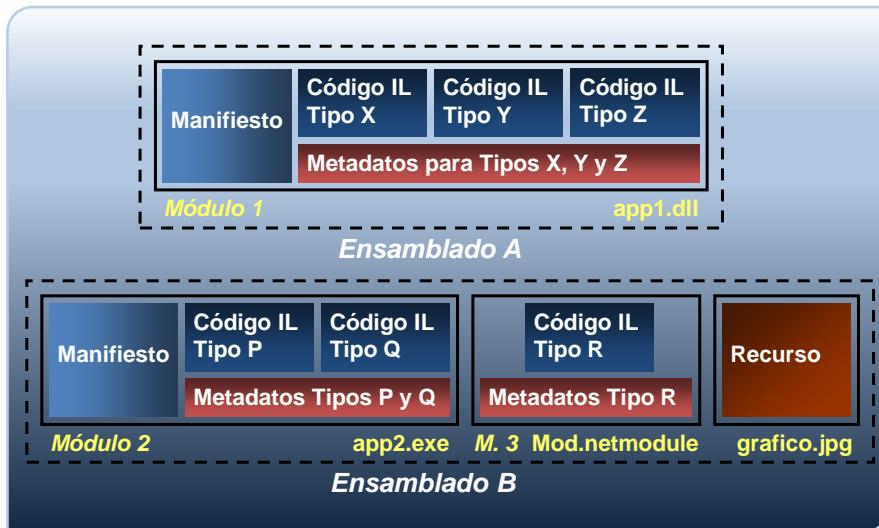
© JMA 2016. All rights reserved

## Proceso del código gestionado



© JMA 2016. All rights reserved

## Anatomía de un ensamblado



© JMA 2016. All rights reserved

## Ensamblados

- Los ensamblados son las unidades de creación de las aplicaciones .NET Framework; constituyen la unidad fundamental de implementación, control de versiones, reutilización, ámbitos de activación (friend o internal) y permisos de seguridad.
- Tipos (según su persistencia):
  - Estáticos: Se almacenan en disco: Ejecutables (EXE) o Librerías (DLL)
  - Dinámicos: Se crean y ejecutan directamente en memoria aunque se pueden guardar una vez ejecutados (Reflection.Emit).

© JMA 2016. All rights reserved

# Ensamblados

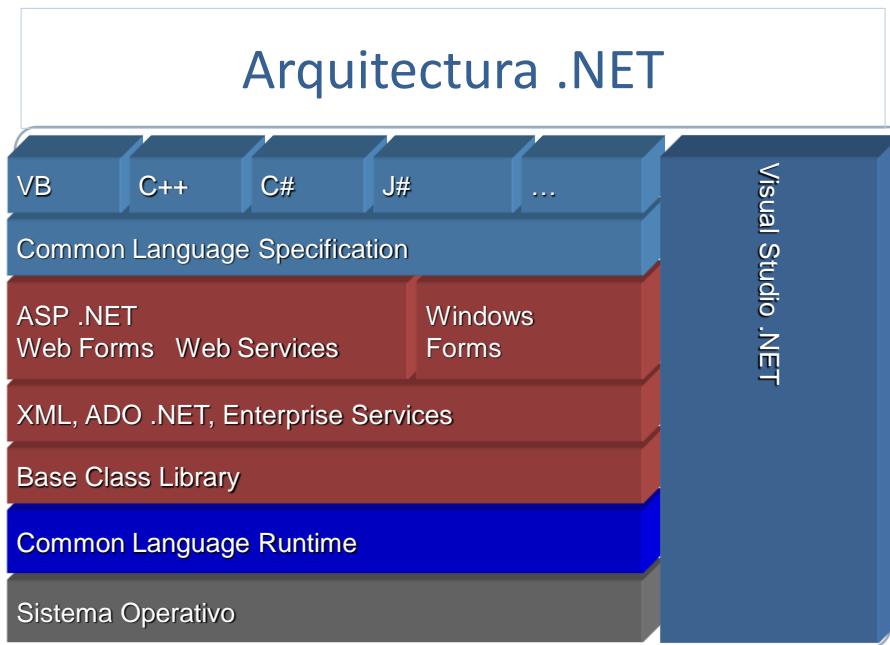
- Tipos (según su visibilidad):
  - Privados: Se almacenan en el directorio de la aplicación o en uno de sus subdirectorios.
  - Compartidos: Se almacenan en el GAC (Caché de ensamblados global). Control de versiones, seguros (firmados con nombre seguro), eficientes (ya verificados y rápida localización).
- Sondeo:
  - Directorio local del ensamblado
    - Caché de ensamblados global
    - Subdirectorios del Directorio local del ensamblado

© JMA 2016. All rights reserved

# Modelos de implementación

- La publicación de una aplicación como **independiente** genera un archivo ejecutable que incluye el entorno de ejecución y las bibliotecas de .NET, así como la aplicación y sus dependencias. Los usuarios de la aplicación pueden ejecutarla en un equipo que no tenga instalado el entorno de ejecución de .NET. Las aplicaciones independientes son específicas de la plataforma y, opcionalmente, se pueden publicar mediante una forma de compilación AOT.
- La publicación de una aplicación como **dependiente del marco** genera un archivo ejecutable y archivos binarios (archivos .dll) que solo incluyen la propia aplicación y sus dependencias. Los usuarios de la aplicación tienen que instalar el entorno de ejecución de .NET por separado. El archivo ejecutable es específico de la plataforma, pero los archivos .dll de las aplicaciones dependientes del marco son multiplataforma. Puede instalar varias versiones del tiempo de ejecución en paralelo para ejecutar aplicaciones dependientes del marco destinadas a otras versiones del tiempo de ejecución.

© JMA 2016. All rights reserved

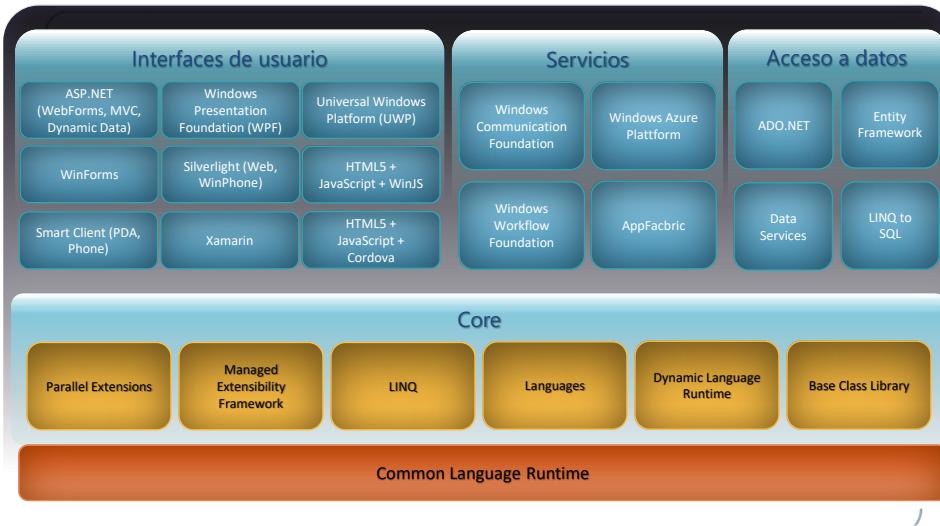


## Tipos de aplicaciones

- Aplicaciones de consola
- Aplicaciones de escritorio
  - Windows WPF
  - Windows Forms
  - Plataforma universal de Windows (UWP)
- Servicios de Windows
- Aplicaciones web, API web y microservicios
- Funciones sin servidor en la nube
- Aplicaciones nativas de la nube
- Aplicaciones móviles
- Juegos
- Internet de las cosas (IoT)
- Aprendizaje automático (Machine Learning)

© JMA 2016. All rights reserved.

# Mapa de tecnologías



© JMA 2016. All rights reserved

## Herramientas de .NET e infraestructura común

- Los lenguajes .NET y sus compiladores
- El sistema de proyectos de .NET (basado en archivos .csproj, .vbproj y .fsproj)
- MSBuild, el motor de compilación usado para compilar proyectos
- NuGet, administrador de paquetes de Microsoft para .NET
- Herramientas de organización de compilación de código abierto, como CAKE y FAKE

© JMA 2016. All rights reserved

## ¿Dónde instalar el .NET Framework?

	<b>Cliente</b>	<b>Servidor</b>
<b>Aplicación de Escritorio</b>	✓	✓*
<b>Aplicación Web</b>		✓
<b>Aplicación de Consola</b>	✓	✓*
<b>Aplicación Móvil</b>	.NET Compact Framework	

**\* Sólo si la aplicación es distribuída**

© JMA 2016. All rights reserved

## Evolución del .NET

- Versiones

Año	CLR	.NET	Visual Studio	C#	VB
2002	1.0	1.0	2002 (7.0)	1.0	7.0
2003	1.1	1.1	2003 (7.1)	1.0	7.0
2005	2.0	2.0	2005 (8.0)	2.0	8.0
2006	2.0**	3.0	.NET 3.0 ext.	2.0**	8.0**
2007	2.0**	3.5	2008 (9.0)*	3.0	9.0
2010	4.0	4.0	2010 (10.0)*	4.0	10.0

\* Posibilidad de compilar aplicaciones en las versiones anteriores (hasta la 2.0)

\*\* Funcionalidad adicionales agregadas a la versión del producto

© JMA 2016. All rights reserved

# Herramientas

- Profesionales:
  - Visual Studio 2022 con la carga de trabajo de “Desarrollo multiplataforma de .NET Core”
- Open source:
  - Visual Studio Code (<https://code.visualstudio.com/>)
  - Extensión C# para Visual Studio Code (<https://marketplace.visualstudio.com/items?itemName=ms-dotnettools.csharp>)
  - SDK de .NET Core 3.0 o versiones posteriores (<https://dotnet.microsoft.com/download>)
  - La interfaz de la línea de comandos (CLI) de .NET Core es una cadena de herramientas multiplataforma para desarrollar, compilar, ejecutar y publicar aplicaciones .NET Core.

© JMA 2016. All rights reserved

## dotnet cli

- El comando dotnet tiene dos funciones:
  - Proporcionar comandos para trabajar con proyectos de .NET: Cada comando define sus propias opciones y argumentos. Todos los comandos admiten la opción --help para ver una breve documentación sobre cómo usar el comando.
  - Ejecuta aplicaciones de .NET.: Hay que especificar la ruta de acceso al archivo .dll de una aplicación para poder ejecutarla. Ejecutar la aplicación significa buscar y ejecutar el punto de entrada, que en el caso de las aplicaciones de consola es el método Main. Por ejemplo, dotnet myapp.dll ejecuta la aplicación myapp.
- dotnet --info

© JMA 2016. All rights reserved

## Comandos de dotnet

<code>dotnet build</code>	Compila una aplicación de .NET.
<code>dotnet build-server</code>	Interactúa con servidores iniciados por una compilación.
<code>dotnet clean</code>	Limpia las salidas de la compilación.
<code>dotnet help</code>	Muestra documentación más detallada en línea sobre el comando.
<code>dotnet migrate</code>	Migra un proyecto de Preview 2 válido a un proyecto del SDK 1.0 de .NET Core.
<code>dotnet msbuild</code>	Proporciona acceso a la línea de comandos de MSBuild.
<code>dotnet new</code>	Inicializa un proyecto de C# o F# para una plantilla determinada.
<code>dotnet pack</code>	Crea un paquete de NuGet de su código.
<code>dotnet publish</code>	Publica una aplicación dependiente del maco .NET o autocontenida.
<code>dotnet restore</code>	Restaura las dependencias de una aplicación determinada.
<code>dotnet run</code>	Ejecuta la aplicación desde el origen.
<code>dotnet sdk check</code>	Muestra el estado actualizado de las versiones instaladas del SDK y el entorno de ejecución.
<code>dotnet sln</code>	Opciones para agregar, quitar y mostrar proyectos en un archivo de solución.
<code>dotnet store</code>	Almacena ensamblados en el almacenamiento de paquetes en tiempo de ejecución.
<code>dotnet test</code>	Ejecuta pruebas usando un ejecutor de pruebas.
<code>dotnet * reference</code>	Agrega (add), quita (remove) o enumera (list) referencias de proyecto.
<code>dotnet * package</code>	Agrega (add), quita (remove) o enumera (list) un paquete NuGet.

© JMA 2016. All rights reserved

## Comandos de dotnet

- Para enumerar las plantillas disponibles:
  - `dotnet new --list`
- Para crear un nuevo proyecto según la plantilla especificada
  - `dotnet new mvc -au Individual -o myWebApp`
- Para confiar en el certificado autofirmado:
  - `dotnet dev-certs https --trust`
- Para eliminar el certificado autofirmado:
  - `dotnet dev-certs https --clean`
- Para compilar y ejecutar en modo continuo:
  - `dotnet watch run`
- Para compilar un proyecto y todas sus dependencias.
  - `dotnet build -c Release --output ./build`
- Para publicar un proyecto y todas sus dependencias.
  - `dotnet publish -c Release --output ./publish`

© JMA 2016. All rights reserved

## CONCEPTOS BÁSICOS

© JMA 2016. All rights reserved.

### Conceptos POO

- Objetos: Clases e instancias
- Miembros: Campos (atributos), métodos, propiedades y eventos
- Constructores y destructores
- Encapsulación, reutilización y Herencia
- Polimorfismo e interfaces
- Sobrecarga, reemplazo y sombreado
- Clases abstractas
- Miembros de clase o compartidos

© JMA 2016. All rights reserved.

# Componentes

- Miembros: propiedades y eventos
- Delegados
- Programación orientada a eventos
  - Emisor
    - Definir EventArg
    - Definir EventHandler
    - Declarar Event
    - Implementar método protegido onEvento(eventArg)
    - Lanzar eventos (invocar onEvento)
  - Receptor
    - Implementar controlador de eventos
    - Asociar controlador de evento al evento

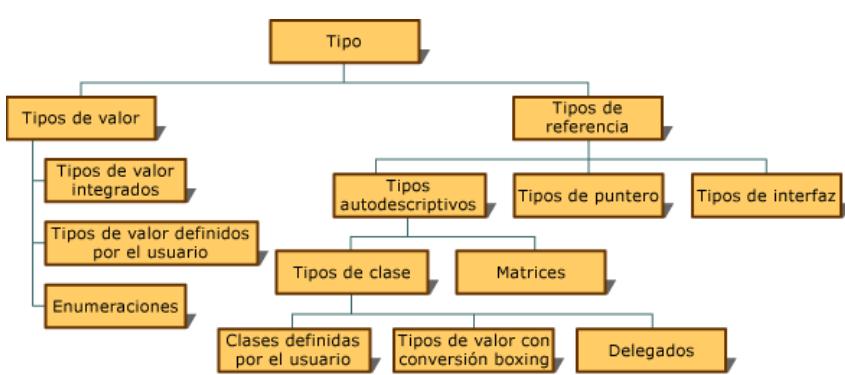
© JMA 2016. All rights reserved

# Conceptos Avanzados

- Genéricos
- Indicadores
- Clases y Métodos Partial
- Métodos extensores
- Declaraciones: inferencia de tipos, dinámicos
- Delegados, métodos anónimos, Expresiones lambda
- Tratamiento de excepciones
- Espacios de Nombres
- Atributos (anotaciones o metadatos)
- Reflexión
- El documentador

© JMA 2016. All rights reserved

# CTS (Sistema de tipos común)



© JMA 2016. All rights reserved.

# Interfaz integrada de desarrollo (IDE)

- Página de inicio
- Menús y barras de botones
- Ventanas acopiables
  - Ocultar automáticamente
  - Explorador de soluciones
  - Vista de clases
  - Explorador de servidores
  - Cuadro de herramientas
  - Propiedades
  - Ayuda dinámica
  - Lista de tareas
- Editor
  - Multi-solapa
  - Formateo avanzado, Refactorización y Generación de código
  - Sensible a contexto, Documentador
  - Diseñadores y vistas
    - Formularios Windows y Web
    - HTML, CSS, JS, XML y esquemas
    - Bases de datos y consultas
    - Componentes y controles de usuarios
    - Gráficos y recursos

© JMA 2016. All rights reserved.

# Depuración

- Modos Debug y Release
  - Puntos de interrupción, condicionales, ...
  - Depuración paso a paso
  - Parar, modificar y continuar.
  - Inspección de variables
  - Traza e IntelliTrace
- Depuración remota
- Análisis de código fuente
- Proyectos de pruebas, rendimientos y diagnósticos
- Clases Debug y Trace

© JMA 2016. All rights reserved

# Soluciones y proyectos

- Soluciones
  - Carpetas
    - Proyectos
      - Referencias
      - Carpetas
      - BIN
      - OBJ
- Proyectos web
- Plantillas de proyectos
- NuGet: Administrador de paquetes
- Control de código fuente (TFS, Git, ...)
- Plantillas T4

© JMA 2016. All rights reserved

# Informes

- Generación de informes:
  - GDI+
  - Reports
  - Crystal Reports .NET
  - Flow Document
- Despliegue (DISTRIBUCIÓN DE APLICACIONES .NET):
  - XCOPY
  - Web
  - ClickOne
  - MS Installer

© JMA 2016. All rights reserved

# LENGUAJE C#

© JMA 2016. All rights reserved

# Introducción

- C# ha sido diseñado específicamente para la plataforma .NET
- Lenguaje orientado a objetos y componentes
- Heredero del C++ y Java, sintaxis común mejorada y ampliada
- Algunas de las principales características de este lenguaje incluyen clases, interfaces, delegados, boxing y unboxing, espacios de nombres, propiedades, indexadores, eventos, sobrecarga de operadores, versionado, atributos, código inseguro, y la creación de documentación en formato XML. No son necesarios archivos de cabecera ni archivos IDL (Interface Definition Language).

© JMA 2016. All rights reserved

# Introducción

- Sencillez
  - Código autocontenido
  - Tamaño de tipos básicos fijo
  - Operador acceso no mutable (., ::)
  - Sin macros ni herencia múltiple
- Modernidad
  - Amplio conjunto de tipos básicos
  - Estructuras inmediatas (foreach)
  - Atributos o metadatos.
  - Excepciones
- Orientación a objetos
  - Encapsulación: public, private, protected, internal
  - Herencia: Sin herencia múltiple. Suplida por interfaces
  - Polimorfismo: Usando herencia o interfaces.
  - Por defecto métodos sellados. Modificador virtual
  - POO pura. Sin métodos o variables globales
- Orientación a componentes
  - Propiedades y Eventos
- Paradigma funcional y lenguajes dinámicos
  - Inferencia de tipos
  - Expresiones Lambda
  - Tipos anónimos, dinámicos,

© JMA 2016. All rights reserved

# Sintaxis

- Sensible a mayúsculas y minúsculas.
- Marca de fin de instrucción: ; (punto y coma): Instrucción en varias líneas sin partir palabras. Pueden ir varias instrucciones en una línea.
- Comentarios: // (hasta fin de línea) o /\* ... \*/ (bloque de comentario)
- Literales:
  - Booleanos: true y false
  - Nulo: null
  - Numéricos: Decimal, Hexadecimal: 0x, Binario: 0b, Exponencial: E
    - Separadores de dígitos ([ver. 7](#)): 0b0001\_0000, 100\_000\_000\_000
  - Caracteres: '...' (Apostrofe)
  - Cadenas: "..." (Comillas) o @..." (cadena textual)
    - Interpolación ([ver. 6](#)): \$"...{expresion}..."
    - Literales de cadena sin formato ([ver. 11](#)): múltiples líneas sin necesidad de secuencias de escape, comienzan con al menos tres caracteres de comillas dobles (""""") y terminan con el mismo número. Varios caracteres \$ indican cuántas llaves consecutivas comienzan y terminan la interpolación.
- Delimitadores de bloques: {...}
- Etiquetas → Nombre\_etiqueta:

© JMA 2016. All rights reserved

## Secuencia de escape

Secuencia de escape	Nombre del carácter
\u...	Unicote
\x...	Hexadecimal
'	Comilla simple
"	Comilla doble
\\	Barra invertida
\0	Null
\a	Alerta
\b	Retroceso
\f	Avance de página
\n	Nueva línea
\r	Retorno de carro
\t	Tabulación horizontal
\v	Tabulación vertical

© JMA 2016. All rights reserved

## Un nombre de elemento o identificadores

- No debe comenzar por un dígito.
- Puede contener caracteres alfabéticos (incluidos acentuados, la ñ y caracteres Unicode no imprimibles), dígitos y signos de subrayado.
- No se pueden utilizar los caracteres de puntuación y símbolos utilizados en el lenguaje.
- No puede superar los 16.383 caracteres de longitud.
- No puede coincidir con ninguna de las palabras clave reservadas si no están precedido por @ (Nombres de escape).
- En la elección debe tenerse en cuenta que influye en los ensamblados reutilizados en otros lenguajes.

© JMA 2016. All rights reserved

## Documentador

- Comentarios XML, comienzan por la marca ///
- Preceden inmediatamente al elemento a comentar.

Etiquetas recomendadas	Finalidad
<summary>	Describe un miembro de un tipo
<c>	Establecer un tipo de fuente de código para un texto
<code>	Establecer una o más líneas de código fuente o indicar el final de un programa
<example>	Indicar un ejemplo
<exception>	Identifica las excepciones que pueden iniciar un método
<include>	Incluye XML procedente de un archivo externo
<list>	Crear una lista o una tabla
<para>	Permite agregar un párrafo al texto
<param>	Describe un parámetro para un método o constructor

© JMA 2016. All rights reserved

# Documentador

Etiquetas recomendadas	Finalidad
<paramref>	Identifica una palabra como nombre de parámetro
<permission>	Documenta la accesibilidad de seguridad de un miembro
<returns>	Describe el valor devuelto de un método
<see>	Especifica un vínculo
<seealso>	Genera una entrada de tipo Vea también
<remarks>	Describe un tipo
<value>	Describe una propiedad
<typeparam>	Describe un parámetro de tipo genérico
<typeparamref>	Identifica una palabra como nombre de parámetro de tipo

- Sandcastle - Documentation Compiler for Managed Class Libraries
- <http://sandcastle.codeplex.com/>

© JMA 2016. All rights reserved

# Directivas al compilador

- Compilación condicional

```
#if <condición1>
    <código1>
#elif <condición2>
    <código2>
#else
    <códigoElse>
#endif
```
- Definición de constantes

```
#define <nombrelidentificador>
#undef <nombrelidentificador>
```

Constantes predefinidas: DEBUG y TRACE  
En VS: Proyectos → Propiedades → Propiedades de configuración  
→ Generar → Constantes de compilación condicional

© JMA 2016. All rights reserved

## Directivas al compilador

- Generación de diagnósticos  
`#warning <Mensaje de Aviso>`  
`#error <Mensaje de Error>`
- Supresión temporal de avisos  
`# pragma warning disable <Número de Error>`  
`// código ...`  
`# pragma warning restore <Número de Error>`
- Cambios en la numeración de líneas  
`#line <número> "<nombreFichero>"`
- Regiones de código  
`#region "Descripción de la Región"`  
`// código ...`  
`#endregion`

© JMA 2016. All rights reserved

## Código fuente

- Estructura del código fuente:
  1. Instrucciones de importación, si corresponde
  2. Instrucciones de Espacio de nombres, si corresponde
  3. Instrucciones class, struct, interface, delegate y enum, si corresponde
- Procedimiento Principal
  - static void Main() {...}
  - static void Main(string[] args) {...}
  - static int Main() {...}
  - static int Main(string[] args) {...}

© JMA 2016. All rights reserved

## Instrucciones de nivel superior (ver: 9.0)

- A partir de C# 9, no es necesario incluir explícitamente un método Main en un proyecto de aplicación de consola. En su lugar, se puede usar la característica de instrucciones de nivel superior para minimizar el código que se tiene que escribir. En este caso, el compilador genera una clase y un punto de entrada con el método Main para la aplicación.
- Una aplicación solo debe tener un punto de entrada. Un proyecto solo puede tener un archivo con instrucciones de nivel superior.

```
// Program.cs
using System.Linq;

if(args.Length > 0) {
    foreach(var arg in args)
        Console.WriteLine($"Argument={arg}");
} else
    Console.WriteLine("Hello, World!");
return 0;
```

- Se puede llamar a un método asincrónico mediante el uso await.

© JMA 2016. All rights reserved

## TIPOS, VARIABLES, CONSTANTES, OPERADORES, INSTRUCCIONES

© JMA 2016. All rights reserved

## Tipos valor

Tipo en C#	Estructura de tipo CLR	Descripción	Almacenamiento nominal	Intervalo de valores
sbyte	System.SByte	Bytes con signo	8 bytes	-128 a 127
byte	System.Byte	Bytes sin signo	8 bytes	0 a 255
short	System.Int16	Enteros cortos con signo	16 bytes	-32.768 a 32.767
ushort	System.UInt16	Enteros cortos sin signo	16 bytes	0 a 65.535
int	System.Int32	Enteros normales	32 bytes	-2.147.483.648 a 2.147.483.647
uint	System.UInt32	Enteros normales sin signo	32 bytes	0 a 4.294.967.295
long	System.Int64	Enteros largos	64 bytes	-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807
ulong	System.UInt64	Enteros largos sin signo	64 bytes	0 a 18.446.744.073.709.551.615

© JMA 2016. All rights reserved

## Tipos valor

Tipo en C#	Estructura de tipo CLR	Descripción	Almacenamiento nominal	Intervalo de valores
float	System.Single	Reales punto flotante con precisión simple	32 bytes	-3.4028235E+38 a -1,401298E-45 para valores negativos; 1,401298E-45 a 3,4028235E+38 para valores positivos.
double	System.Double	Reales punto flotante con precisión doble	64 bytes	-4,94065645841246544E-324 a 4,94065645841246544E-324
decimal	System.Decimal	Reales de 28 dígitos decimales de precisión	128 bytes	35 con 28 posiciones a la derecha del signo decimal; (+/-1E-28)
bool	System.Boolean	Valores lógicos	32 bytes	true o false
char	System.Char	Caracteres Unicode	16 bytes	0 a 65535 (sin signo) ['\u0000', '\uFFFF']
string	System.String	Cadena de longitud variable	En función a la plataforma	De 0 a 2.000 millones de caracteres Unicode aprox.
object	System.Object	Cualquier objeto	4 bytes	Cualquier objeto

© JMA 2016. All rights reserved

## Sufijos de constantes numéricas

Sufijo	Tipo del literal entero
ninguno	Primero de: <b>int, uint, long, ulong</b>
<b>L ó l</b>	Primero de: <b>long, ulong</b>
<b>U ó u</b>	Primero de: <b>int, uint</b>
<b>UL, UI, uL, ul, LU, Lu, IU ó lu</b>	<b>ulong</b>

Sufijo	Tipo del literal real
<b>F ó f</b>	<b>float</b>
ninguno, <b>D ó d</b>	<b>double</b>
<b>M ó m</b>	<b>decimal</b>

© JMA 2016. All rights reserved

## Tipos valor como referencias

- **Boxing y Unboxing (Empaquetado y desempaquetado)**
  - Los tipos valor se empaquetan en sus correspondientes clases.
  - Permite a los tipos valor comportarse como objetos.
  - Aportan funcionalidad adicional al tipo valor
  - Mantienen la eficiencia de los tipos valor.
- **Tipos valor que aceptan NULL**
  - Los tipos valor se transforman en sus correspondientes clases Nullable.
  - Se crean utilizando el sufijo modificador de tipo ?
  - Tiene dos propiedades públicas de sólo lectura: HasValue, del tipo bool, y Value, del tipo subyacente del tipo que acepta valores NULL.

© JMA 2016. All rights reserved

## Tipos de referencia

Tipo de clase	Descripción
System.Object	Clase base definitiva de todos los demás tipos.
System.String	Tipo de cadena del lenguaje C#.
System.ValueType	Clase base de todos los tipos de valor.
System.Enum	Clase base de todos los tipos enum.
System.Array	Clase base de todos los tipos de matriz.
System.Delegate	Clase base de todos los tipos delegados.
System.Exception	Clase base de todos los tipos de excepción.

© JMA 2016. All rights reserved

## Conversiones

- **Implícitas**
  - Valor: de un tipo otro de mayor rango de valores.
  - Referencias: de un tipo derivado a cualquiera de sus tipos superiores en la jerarquía de herencia y tipo que implementa un interfaz a un tipo de la interfaz implementada.
- **Explícitas**
  - Empleando funciones o métodos de conversión (System.Convert).
  - Mediante la sobrecarga de operadores.
  - (**<TipoResultante><Referencia>** ó **<Referencia> as <TipoResultante>**)
    - Siempre y cuando el tipo declarado de la referencia sea un tipo superior en la jerarquía de herencia del tipo resultante o el tipo declarado de la referencia sea de un tipo interfaz que implemente el tipo resultante. Requiere comprobación de tipos (operador is).

© JMA 2016. All rights reserved

# Expresiones y Operadores

Operador	Descripción
<code>&lt;objeto&gt;.&lt;miembro&gt;</code>	Accede a miembros de los objetos o estructuras.
<code>&lt;nombre&gt;[&lt;índice&gt;]</code>	Acceso a los elementos de las matrices e indizadores.
<code>[&lt;Metadatos&gt;]</code>	Marca un bloque de atributos.
<code>&lt; ... &gt;</code>	Marca un bloque de parámetros tipo en los genéricos.
<code>(&lt;expresión&gt;)</code>	Prioriza la evaluación de la expresión.
<code>&lt;nombre&gt;(...)</code>	Invocación de métodos y delegados.
<code>new &lt;tipo&gt;</code>	Se utiliza para crear objetos e invocar constructores
<code>new { ... }</code>	Inicializador de objeto anónimo.
<code>typeof(&lt;tipo&gt;)</code>	Obtiene el objeto System.Type para un tipo.
<code>default(&lt;tipo&gt;)</code>	Obtiene el valor predeterminado del tipo.
<code>checked(&lt;expresión&gt;)</code>	Evaluúa la expresión en contexto comprobado
<code>unchecked(&lt;expresión&gt;)</code>	Evaluúa la expresión en contexto no comprobado

© JMA 2016. All rights reserved

# Expresiones y Operadores

Operador	Descripción
<code>&lt;expr1&gt; + &lt;expr2&gt;</code>	Suma operandos aritméticos. Concatenación de cadenas. Combinación de delegados.
<code>&lt;expr1&gt; - &lt;expr2&gt;</code>	Resta operandos aritméticos. Elimina la asociación de delegados.
<code>&lt;expr1&gt; * &lt;expr2&gt;</code>	Multiplica operandos aritméticos.
<code>&lt;expr1&gt; / &lt;expr2&gt;</code>	Divide operandos aritméticos. El resultado será real o entero en función de los operandos.
<code>&lt;expr1&gt; % &lt;expr2&gt;</code>	Modulo (resto) de la división entera.
<code>++&lt;variable&gt;</code>	Preincremento: incrementa en 1 el valor de la variable antes de consultar su valor.
<code>&lt;variable&gt;++</code>	Postincremento: incrementa en 1 el valor de la variable después de consultar su valor.
<code>- &lt;variable&gt;</code>	Predecremento: decrementa en 1 el valor de la variable antes de consultar su valor.
<code>&lt;variable&gt;- -</code>	Postdecremento: decrementa en 1 el valor de la variable después de consultar su valor.

© JMA 2016. All rights reserved

# Expresiones y Operadores

Operador	Descripción
<code>&lt;expr1&gt; &amp; &lt;expr2&gt;</code>	AND binario para operadores de tipos enteros y lógico para operadores booleanos.
<code>&lt;expr1&gt;   &lt;expr2&gt;</code>	OR binario para operadores de tipos enteros y lógico para operadores booleanos.
<code>&lt;expr1&gt; ^ &lt;expr2&gt;</code>	XOR binario para operadores de tipos enteros y lógico para operadores booleanos.
<code>&lt;expr&gt; &lt;&lt; &lt;contador&gt;</code>	Desplaza a la izquierda sin desbordamiento los bit de la expresión rellenado con ceros las posiciones libres.
<code>&lt;exprn&gt; &gt;&gt; &lt;contador&gt;</code>	Desplaza a la derecha sin desbordamiento los bit de la expresión rellenado con ceros las posiciones libres.
<code>~&lt;expresión&gt;</code>	Complemento binario. Negación bit a bit.
<code>!&lt;expresión&gt;</code>	Negación lógica.
<code>&lt;expr1&gt; &amp;&amp; &lt;expr2&gt;</code>	AND lógico cortocircuitado, solo evalúa la segunda expresión si la primera es true.
<code>&lt;expr1&gt;    &lt;expr2&gt;</code>	OR lógico cortocircuitado, solo evalúa la segunda expresión si la primera es false.

© JMA 2016. All rights reserved

# Expresiones y Operadores

Operador	Descripción
<code>&lt;expr1&gt;&lt;comp&gt; &lt;expr2&gt;</code>	Compara dos expresiones, donde <code>&lt;comp&gt;</code> puede ser: <code>==</code> (igual), <code>!=</code> (distinto), <code>&lt;</code> (menor), <code>&lt;=</code> (menor o igual), <code>&gt;</code> (mayor), <code>&gt;=</code> (mayor o igual).
<code>&lt;expresión&gt; is &lt;tipo&gt;</code>	Comprueba si el tipo en tiempo de ejecución de un objeto es compatible con un tipo dado
<code>&lt;cond&gt;?&lt;exp1&gt;:&lt;exp2&gt;</code>	Condicional: devuelve el primer valor sin la condición es true o el segundo si es false.
<code>&lt;objeto&gt;?.&lt;miembro&gt;</code> <code>&lt;objeto&gt;? [&lt;índice&gt;]</code>	Condicional de NULL ( <b>ver. 6.0</b> ): si el objeto es nulo devuelve nulo, sino el resultado de invocar la miembro, equivale a <code>&lt;objeto&gt; == null ? null : &lt;objeto&gt;.&lt;miembro&gt;</code>
<code>&lt;expr1&gt;??&lt;expr2&gt;</code>	Condicional: devuelve el primer valor si no es nulo o el segundo si el primero es nulo. (Valor de sustitución del null)
<code>&lt;expresión&gt; as &lt;tipo&gt;</code>	Devuelve conversión de referencias entre tipos compatibles o null si no es compatible.
<code>(&lt;tipo&gt;) &lt;expresión&gt;</code>	Realiza conversiones explícitas entre tipos compatibles.

© JMA 2016. All rights reserved

# Expresiones y Operadores

Operador	Descripción
<code>&lt;dest&gt; = &lt;expr&gt;</code>	Asigna la expresión al destinatario.
<code>&lt;dest&gt; &lt;opr&gt;= &lt;expr&gt;</code>	Equivale a <code>&lt;destinatario&gt; = &lt;destinatario&gt; &lt;opr&gt; &lt;expr&gt;</code> Donde el <code>&lt;opr&gt;</code> puede ser: * / % + - << >> & ^
<code>( ... ) =&gt; &lt;expresión&gt;</code>	Función anónima (expresión lambda) ( <b>ver. 4.0</b> )
<code>delegate { ... }</code>	Función anónima (método anónimo)
<code>nameof(&lt;miembro&gt;)</code>	<b>(ver. 6.0)</b> Extrae una cadena con el identificador de la variable, propiedad o campo especificado como argumento (durante la compilación con comprobación de tipos): <code>OnPropertyChanged(nameof(Propiedad));</code>
<code>&amp;&lt;expresión&gt;</code>	Dirección de memoria de su operando. [No administrado].
<code>&lt;expr1&gt;-&gt;&lt;expr2&gt;</code>	Acceso a un miembro de la estructura apuntada. El primer operador debe ser un puntero. Similar a <code>(*&lt;expr1&gt;).&lt;expr2&gt;</code> . [No administrado].
<code>sizeof(&lt;tipo&gt;)</code>	Obtiene el tamaño en bytes de un tipo de valor.
<code>stackalloc &lt;tipo&gt;[&lt;tamaño&gt;]</code>	Asigna un bloque de memoria en la pila y devuelve un puntero. [No administrado].

© JMA 2016. All rights reserved

# Expresiones y Operadores

Operador	Descripción
<code>&lt;exp&gt; switch {&lt;patrón&gt;}</code>	<b>(ver.7)</b> Para evaluar una expresión única a partir de una lista de expresiones candidatas basadas en una coincidencia de patrón con una expresión de entrada.
<code>&lt;obj&gt; with {&lt;inicializador&gt;}</code>	<b>(ver.9)</b> Genera una copia de su operando con las propiedades y campos especificados modificados
<code>&lt;alias&gt;::&lt;tipo&gt;</code>	Calificador para acceder a un miembro del espacio de nombres con alias.
<code>await &lt;tarea&gt;</code>	Suspende la evaluación del método <code>async</code> envolvente hasta que se completa la operación asíncrona representada por su operando.
<code>.. col1</code>	<b>(ver.12)</b> El operador de propagación, .. en una expresión de colección reemplaza su argumento por los elementos de esa colección o array.

© JMA 2016. All rights reserved

# Expresiones y Operadores

Categoría	Prioridad de Operadores
Principal	(Expresión) x.y método(x) tabla[x] x++ x-- new typeof checked unchecked
Unario	+ - ! ~ ++x --x (T)x
Multiplicativo	* / %
Sumatorio	+ -
Desplazamiento	<< >>
Tipos y relacionales	< > <= >= is as
Igualdad	== !=
AND lógico	&
XOR lógico	^
OR lógico	
AND condicional	&&
OR condicional	
Uso combinado de Null	??
Condicional	?:
Asignación	= *= /= %= += -= <<= >>= &= ^=  = (ver. 8.0) ??= /

© JMA 2016. All rights reserved

## Detección de patrones (ver: 7.0)

- La coincidencia de patrones es una característica que permite implementar la distribución de métodos en propiedades distintas al tipo de un objeto.
- Permite crear variables al vuelo una vez detectado el tipo evitando casting o asignaciones explícitas.
- La coincidencia de patrones admite expresiones is y switch. Cada una de ellas habilita la inspección de un objeto y sus propiedades para determinar si el objeto cumple el patrón buscado.

```
if (shape is Square s)
    return s.Side * s.Side;
else if (shape is Circle c)
    return c.Radius * c.Radius * Math.PI;
switch (shape) {
    case Square s: return s.Side * s.Side;
    case Circle c: return c.Radius * c.Radius * Math.PI;
```

- Con la palabra clave when se puede especificar reglas adicionales para el patrón.

```
switch (shape) {
    case Square s when s.Side == 0:
    case Circle c when c.Radius == 0:
        return 0;
    case Square s: return s.Side * s.Side;
```

© JMA 2016. All rights reserved

## Coincidencia de patrones (ver: 8.0)

- La coincidencia de patrones ofrece herramientas que proporcionan funcionalidad dependiente de la forma entre tipos de datos relacionados pero diferentes.

```
public static RGBColor FromRainbow(Rainbow colorBand) =>
    colorBand switch {
        Rainbow.Red => new RGBColor(0xFF, 0x00, 0x00),
        Rainbow.Green => new RGBColor(0x00, 0xFF, 0x00),
        Rainbow.Blue => new RGBColor(0x00, 0x00, 0xFF),
        _ => throw new ArgumentException(message: "invalid value", paramName:
            nameof(colorBand)),
    };
```

- El patrón de propiedades permite hallar la coincidencia con las propiedades del objeto examinado.

```
segment is { Start.Y: 0 } or { End.Y: 0 };
location switch {
    { State: "WA" } => salePrice * 0.06M,
```

- Los patrones de tupla permiten hacer cambios en función de varios valores, expresados como una tupla.

```
public static string RockPaperScissors(string first, string second) =>
    (first, second) switch {
        ("rock", "paper") => "rock is covered by paper. Paper wins.",
```

© JMA 2016. All rights reserved

## Patrones lógicos (ver: 9.0)

- A partir de C# 9.0, se usan los combinadores de patrones not, and y or para crear los patrones lógicos.

```
static bool IsLetter(char c) => c is (>= 'a' and <= 'z') or (>= 'A' and <= 'Z');
static string Classify(double measurement) => measurement switch {
    < -40.0 => "Too low",
    >= -40.0 and < 0 => "Low",
    >= 0 and < 10.0 => "Acceptable",
    >= 10.0 and < 20.0 => "High",
    >= 20.0 => "Too high",
    double.NaN => "Unknown",
};
static string GetCalendarSeason(DateTime date) => date.Month switch {
    3 or 4 or 5 => "spring", 6 or 7 or 8 => "summer",
    9 or 10 or 11 => "autumn", 12 or 1 or 2 => "winter",
    _ => throw new ArgumentOutOfRangeException(nameof(date), $"Date with
unexpected month: {date.Month}."),
};
```

© JMA 2016. All rights reserved

# Variables y Constantes

- Variables

<tipoVariable> <nombreVariable> = <valorInicial>, <nombreVariable> = <valorInicial>, ...;

- Inicialización por defecto por el CLR:

- Numéricas = 0, Booleanas = false, Caracteres = '\x0000', Cadenas = "", Referencias = null

- Expresión de valor predeterminada:

<nombreVariable> = default(<tipo>);

- El ámbito de la variable es el bloque donde está definida.

- Tipos especiales de declaraciones de tipo (3.0):

- var: tipo implícito (ver. 3.0)

- dynamic: omite la comprobación de tipos en tiempo de compilación (ver. 4.0)

- omitir el tipo conocido en una expresión new (ver. 9.0).

- Tipo v = new();

- Constantes

const <tipo> <nombre> = <valor>, <tipo> <nombre> = <valor>, ...;

readonly <tipo> <nombre> = <valor>, <tipo> <nombre> = <valor>, ...;

© JMA 2016. All rights reserved

# Matrices y Colecciones

- Matrices (clase Array<>)

<tipoDatos>[] <nombreTabla>;

<tipoDatos>[] <nombreTabla> = new <tipoDatos>[<númeroDeElementos>];

<tipoDatos>[] <nombreTabla> = new <tipoDatos>[] {<valores>};

<tipoDatos>[] <nombreTabla> = {<valores>};

<tipo>[, ... ] <nombre> = new <tipoDatos>[<númeroDeElementos>,<númeroDeElementos>, ... ];

<tipoDatos>[][]... <nombreTabla>;

- Acceso: Índice 0 a númeroDeElementos-1

<nombreTabla>[<índice>,<índice>] = <Valor>

<Variable> = <nombreTabla>[<índice>,<índice>]

- (ver. 8.0): operador ^ desde el final del índice y operador .. de intervalo:

- ultimo = tabla[^1];

- rango = tabla[0..4];

- Colecciones

<tipoColeccion> <nombreColeccion>;

<tipoColeccion> <nombreColeccion>= new <tipoColeccion>();

<tipoColeccion> <nombreColeccion>= new <tipoColeccion>(<tamañoInicial>)

<tipoColeccion> <nombreColeccion>= new <tipoColeccion>(<otraColeccion>);

<tipoColeccion> <nombreColeccion>= new <tipoColeccion> {<valores>} (ver. 3.0)

© JMA 2016. All rights reserved

## Deconstrucción y descartes (ver: 7.0)

- La deconstrucción proporciona una manera ligera de recuperar varios valores para asignarlos a varias variables o argumentos.  

```
var (id, name) = named;
(int id, string name) = named;
```
- Aunque la deconstrucción apareció junto con las tuplas también pueden usarse con clases, estructuras o interfaces.
- Los descartes son variables de solo escritura cuyos valores se decide omitir, suelen designarse mediante un carácter de guion bajo ("\_") en una asignación.  

```
var (_, name) = named;
```
- Los tipos que no son tuplas no ofrecen compatibilidad integrada con los descartes. Para ello debe implementar de uno o varios métodos Deconstruct de instancia o implementar uno o varios métodos de extensión Deconstruct que devuelvan los valores que le interesen. El método no devuelve ningún valor, y cada valor que se va a deconstruir se indica mediante un parámetro out en la firma del método.  

```
public void Deconstruct(out int id, out string name) { ... }
var (id, name) = obj;
```

© JMA 2016. All rights reserved

## Referencias no nulas (ver: 8.0)

- Una de las novedades de C# 8.0 son los tipos de referencia que aceptan valores NULL y los tipos de referencia que no aceptan valores NULL. El compilador aplica reglas que garantizan que sea seguro desreferenciar dichas variables sin comprobar primero que no se trata de un valor NULL.
- Cualquier tipo de referencia puede tener una de cuatro nulabilidades, que describen cuándo se generan las advertencias:
  - Nonnullable: no se pueden asignar valores NULL a las variables de este tipo. No es necesario comprobar si estas tienen un valor NULL antes de desreferenciarlas (acceder a uno de sus miembros mediante el operador .).
  - Nullable: se pueden asignar valores NULL a las variables de este tipo. Si se desreferencian sin comprobar primero la existencia de valores null, se producirá una advertencia.
  - Oblivious: se trata del estado previo a C# 8.0. Las variables de este tipo se pueden desreferenciar o asignar sin advertencias.
  - Unknown: este es generalmente el caso de los parámetros de tipo en los que las restricciones no indican al compilador que el tipo debe aceptar valores NULL o no aceptar valores NULL.
- La nulabilidad de un tipo en una declaración de variable se controla mediante el contexto en el que se declara la variable.

© JMA 2016. All rights reserved

## Referencia no nulas (ver: 8.0)

- Tanto el contexto de anotación como el de advertencia pueden establecerse para un proyecto en el archivo .csproj para configurar la forma en la que el compilador interpreta la nulabilidad de los tipos y las advertencias que se generan:
  - enable: el contexto de anotación y el contexto de advertencia es enabled. Las variables de un tipo de referencia, como string, no aceptan valores NULL. Todas las advertencias de nulabilidad están habilitadas.
  - warnings: el contexto de anotación es disabled y el contexto de advertencia es enabled. Las variables de un tipo de referencia son inconscientes. Todas las advertencias de nulabilidad están habilitadas.
  - annotations: el contexto de anotación es enabled y el contexto de advertencia es disabled. Las variables de un tipo de referencia, como cadena, no admiten un valor NULL. Todas las advertencias de nulabilidad están deshabilitadas.
  - disable: el contexto de anotación y el contexto de advertencia es disabled. Las variables de tipo de referencia son inconscientes, como en versiones anteriores de C#. Todas las advertencias de nulabilidad están deshabilitadas.

<Nullable>enable</Nullable>

© JMA 2016. All rights reserved

## Referencia no nulas (ver: 8.0)

- En un contexto de anotación que no acepta valores NULL, el carácter ? junto a un tipo de referencia declara un tipo de referencia que acepta valores NULL.  
`public string? RequestId { get; set; }`
- También puede usar directivas para establecer los contextos en cualquier lugar del proyecto (enable, disable, restore, enable warnings, disable warnings, restore warnings, enable annotations, disable annotations, restore annotations):

```
#nullable enable
public class NewsStoryViewModel {
    public DateTimeOffset Published { get; set; }
    public string Title { get; set; }
    public string Uri { get; set; }
}
#nullable restore
```

© JMA 2016. All rights reserved

# Instrucciones condicionales

```

if (expression)
    statement1
[else
    statement2]

switch (expression) {
    case constant-expression:
        statement
        jump-statement
    [default:
        statement
        jump-statement]
}
jump-statement:
    break;
    goto identifier;
    goto case constant-expression;
    goto default;

```

En C# 6 y versiones anteriores, la expresión del switch debe ser una expresión que devuelva un valor de los siguientes tipos:

- Un carácter.
- Una cadena.
- Un booleano.
- Un valor entero, como int o long.
- Un valor enum.

A partir de C# 7.0, la expresión de coincidencia puede ser cualquier expresión que no sea nula.

© JMA 2016. All rights reserved

# Tratamientos de excepciones

```

try {
    try-block
} catch (exception-declaration-1) { ...
} catch (exception-declaration-2) { ...
...
} catch (exception-declaration-n) { ...
} finally { finally-block }

```

Filtrado de excepciones (**ver 6.0**):

catch (exception-declaration-1) when (expression)

throw <objetoExcepciónALanzar>;

Expresiones throw (**ver 7.0**):

name = value ?? throw ...;

© JMA 2016. All rights reserved

## Instrucciones iterativas

- `while (expression) { ... }`
- `do { ... } while (expression);`
- `for ([initializers]; [expression]; [iterators]) { ... }`
- `foreach (type identifier in expression) { ... }`
  - (`System.Collections.IEnumerable`)
- Alteración del flujo:
  - `continue;`
  - `break;`

© JMA 2016. All rights reserved

## Instrucciones de control

- El desbordamiento aritmético produce una excepción.  
`checked { ... }`  
`checked (expression)`
- Se hace caso omiso del desbordamiento aritmético y el resultado se trunca.  
`unchecked { ... }`  
`unchecked (expression)`
- Impide que el recolector de elementos no utilizados cambie la ubicación de una variable. [No administrado].  
`fixed ( type* ptr = expr ) { ... }`
- Bloqueo de exclusión mutua de un objeto, hace esperar al otro subproceso hasta que termine el subproceso.  
`lock(expression) { ... }`

© JMA 2016. All rights reserved

# Instrucciones de control

- La instrucción `using` define un ámbito al final del cual el objeto se destruye. Realiza una invocación implícita del método `Dispose` al terminar el ámbito del bloque, por lo tanto debe implementar la interfaz `System.IDisposable`.
   
`using (expression | type identifier = initializer) { ... }`  
**(ver 8.0):** `using var identifier = new type(); // hasta fin de bloque`
- Salto incondicional (dentro del ámbito de método actual)  
`label-identifier:`  
`goto label-identifier;`
- Devolución del control y de valores:  
`return <valor>;` devuelve el valor de un método  
`yield return <valor>;` genera y devuelve el siguiente valor de una iteración  
`yield break;` indica que se completó la iteración

© JMA 2016. All rights reserved

# Atributos (anotaciones o metadatos)

- Información adicional para el ensamblado, módulo y elementos (y sus miembros)  
`[<indicadorElemento>:<nombreAtributo> (<parámetros>)]`  
 Elemento
  - `assembly`: Indica que el atributo se aplica al ensamblado en que se compile el código fuente que lo contenga.
  - `module`: Indica que el atributo se aplica al módulo en que se compile el código fuente que lo contenga.
  - `<elementos>`: Cualifica al elemento al que precede y no es necesario dar el indicador de elemento.
- Parámetros
  - Parámetros sin nombre: Dependerá de su posición a la hora de determinar a qué parámetro se le está dando cada valor.
  - Parámetros con nombre: Son opcionales y pueden colocarse en cualquier posición en la lista de `<parámetros>` del atributo `<nombreParámetro>=<valor>`.
- El atributo `Obsolete` se utiliza para marcar tipos y miembros de tipos que ya no se deberían utilizar.
- Clases de atributo condicional  
`[Conditional("DEBUG")]`  
`public class TestAttribute : Attribute {}`  
`[Test]`  
`class C {}`

© JMA 2016. All rights reserved

## DEFINICIÓN DE TIPOS

© JMA 2016. All rights reserved

### Modificadores de Accesibilidad

Modificador	Tipo	Descripción
public	Publico	Accesible desde cualquier punto sin restricciones.
protected	Protegido	Accesible solamente desde los miembros del tipo donde se encuentra declarado y desde sus herederos.
internal	Interno o amigo	Accesible solamente desde el ensamblado donde se encuentra declarado.
internal protected	Interno y Protegido	Accesible solamente desde los miembros del tipo donde se encuentra declarado y desde sus herederos declarados en su mismo ensamblado.
private	Privado	Accesible solamente desde los miembros del tipo donde se encuentra declarado.

© JMA 2016. All rights reserved

## Enumeraciones

---

```
[attributes] [modifiers] enum
    <nombreEnumeración> [: <integral-type>]
{
    <literal>[ = <valor>],
    <literal>[ = <valor>],
    ...
}
```

<nombreEnumeración> <nombreVariable> =
 <nombreEnumeración>.<literal>

---

© JMA 2016. All rights reserved.

## Interfaces

---

```
[attributes] [modifiers] interface INombre
    [;<ListaDeInterfces>] {
        Métodos
        Propiedades
        Indizadores
        Eventos
    }[;]
```

Los modificadores permitidos son new y los cinco modificadores de acceso.

Por convenio deberían ir prefijados por la letra I (de Interface) para diferenciarlos de la clase base al realizar las implementaciones.

© JMA 2016. All rights reserved.

# Clases

```
[attributes] [modifiers] class nombre [:base-list] {
    Constructores
    Destructores
    Constantes
    Campos
    Métodos
    Propiedades
    Indizadores
    Operadores
    Eventos
    Delegados
    Clases
    Interfaces
    Estructuras
}[;]
```

## Modificadores de Accesibilidad

Los modificadores **new**, **protected** y **private** sólo se permiten en clases anidadas.

Notación Pascal

© JMA 2016. All rights reserved

# Clases

Modificador	Tipo	Descripción
<b>new</b>	Ocultación o sombreado	Ocultar o sombra un elemento heredado de una clase base.
<b>abstract</b>	Abstracta	Marca la clase como abstracta, no es instanciable. Incompatible con sealed
<b>sealed</b>	Sellada	No se puede heredar de esta clase.

- Modificador parte de la clase
  - El modificador partial indica que la clase puede estar repartida en varios archivos.
- Herencia e interfaces:
  - : <ClaseBase>
  - : <ListaDeInterfaces>
  - : <ClaseBase>, <ListaDeInterfaces>

© JMA 2016. All rights reserved

# Estructuras

---

```
[atributes] [modifiers] struct <Nombre> [: <Lista de interfaces>] {
    Constantes
    Atributos
    Métodos
    Propiedades
    Eventos
    Indizadores
    Operadores
    Constructores
    Constructores estáticos
    Tipos anidados
}
```

Los modificadores permitidos son new y los cinco modificadores de acceso.

© JMA 2016. All rights reserved

# Estructuras

- 
- Similares a las clases, pueden tener: atributos, métodos, propiedades, ...
  - Pero son más rápidas por ser de **tipo valor**.
  - Las estructuras se diferencian de las clases de diferentes maneras:
    - Las estructuras son tipos de valor.
    - Todos los tipos de estructura se heredan implícitamente de la clase System.ValueType.
    - No se puede heredar de una estructura (son implícitamente sealed)
    - La asignación a una variable de un tipo de estructura crea una copia del valor que se asigne.
    - El valor predeterminado de una estructura es el valor producido al establecer todos los campos de tipos de valor en su valor predeterminado, y todos los campos de tipos de referencia en null.
    - Las operaciones boxing y unboxing se utilizan para realizar la conversión entre un tipo struct y un tipo object.
    - El significado de this es diferente para las estructuras (solo ámbito).
    - Las declaraciones de atributos de instancia para una estructura no pueden incluir inicializadores de variable (antes de la **ver. 10.0**).
    - Los atributos de tipo tabla pueden incluir el modificado fixed para indicar búferes fijos (solo en contextos no seguros).
    - Una estructura no puede declarar un constructor de instancia sin parámetros (solo a partir de la **ver. 6.0**).
    - El constructor debe inicializar todos los campos de instancia del tipo (antes de la **ver. 11.0**).
    - Una estructura no puede declarar un destructor.

© JMA 2016. All rights reserved

## Tuplas (ver: 7.0)

- Las tuplas de C# son tipos que permiten agrupar varios valores como una unidad mediante una sintaxis ligera. Entre otras ventajas, incluyen una sintaxis más sencilla, reglas para conversiones en función de un número (denominadas cardinalidad) y tipos de elementos y reglas coherentes para copias, pruebas de igualdad y asignaciones. Los tipos de tupla son tipos valor mutables.

```
var unnamed = ("one", "two"); // (string, string)
var named = (id: 1, name: "one"); // (int, string)
```

- El struct ValueTuple incluye campos denominados Item1, Item2, Item3, etc., similares a las propiedades definidas en los tipos Tuple existentes. Estos nombres son los únicos que se pueden usar en tuplas sin nombre:

```
var rslt = unnamed.Item2; // two
var rslt = named.name; // one
```

- Las tuplas pueden ser tipos de retorno o de parámetros:

```
(int, string) método((string, string) arg) { ... }
```

© JMA 2016. All rights reserved

## Registros (ver.9.0)

- Los tipos de registro son tipos de referencia que son **inmutables** de forma predeterminada.
  - La igualdad se basa en valores e incluye una comprobación de coincidencia de los tipos.
  - Los registros tienen una representación de cadena coherente que se genera de forma automática.
  - Un registro puede heredar de otro registro. Sin embargo, un registro no puede heredar de una clase, y una clase no puede heredar de un registro.
  - Los registros admiten la construcción de copias. La construcción de copias correctas debe incluir las jerarquías de herencia y las propiedades agregadas por los desarrolladores.
  - Los registros se pueden copiar con modificaciones. Estas operaciones de copia y modificación admiten la mutación no destructiva.

© JMA 2016. All rights reserved

## Registros (ver.9.0)

- Cuando se declara un constructor principal en un registro, el compilador genera propiedades públicas de solo lectura de los parámetros del constructor principal. La declaración:
 

```
public record Person(string FirstName, string LastName);
public record struct Point(double X, double Y, double Z); (ver: 10)
public readonly record struct Point(double X, double Y, double Z); (ver: 10)
```
- Equivale a:
 

```
public class Person {
    public string LastName { get; init; } = default!;
    public string FirstName { get; init; } = default!;
    public Person(string first, string last) => (FirstName, LastName) = (first, last);
}
```
- Sintaxis posicional para la definición de propiedad:
 

```
Person person = new("Nancy", "Davolio");
```
- Mutación no destructiva (copia) con la expresión with:
 

```
Person person1 = new("Nancy", "Davolio");
Person person2 = person1 with { FirstName = "John" };
```

© JMA 2016. All rights reserved

## Espacios de nombres

- El espacio de nombres es un ámbito que permite organizar el código y proporciona una forma de crear tipos únicos globalmente exclusivos.
 

```
namespace CompanyName.TechologyName[.Feature][...][.Design] {
    Otro espacio de nombres
    class, struct, interface, delegate y enum, si corresponde
}
```
- No existe una correspondencia 1 a 1 entre espacios de nombres y ensamblados.
- Las clases de un espacio de nombres se pueden repartir en varios ensamblados.
- Un ensamblado puede contener clases de varios espacios de nombres.
- Las declaraciones de espacio de nombres con ámbito de archivo permiten declarar que todos los tipos de un archivo están en un único espacio de nombres. **(ver. 10.0)**

```
namespace CompanyName.TechologyName;
```

© JMA 2016. All rights reserved

# Espacios de nombres

- La directiva `using` se utiliza para:
    - Crear un alias para un espacio de nombres.
    - Permitir el uso de los tipos de un espacio de nombres sin que sea necesario preceder el nombre del tipo por el nombre de su espacio de nombres.
    - El **calificador de alias de espacios de nombres** :: garantiza que las búsquedas de nombres de tipos no se vean afectadas por la introducción de nuevos tipos y miembros.
    - Los alias extern permiten crear y hacer referencia a diferentes jerarquías de espacios de nombres en diferentes ensamblados (se definen como directivas de compilación)
- `using [alias = ]class_or_namespace;`
- Para utilizar una clase de un espacio de nombres, no basta con la directiva `using`, es necesario tener referenciado el ensamblado que la contiene.
  - La directiva `using static (ver. 6.0)` permite especificar una clase en la que se pueden acceder los métodos estáticos disponibles en el ámbito global, sin prefijos de tipo.  
`using static System.Math;`
  - El modificador `global using (ver. 10.0)` hará que el `using` se aplique a todos los archivos de la compilación (normalmente un proyecto), puede aparecer al principio (antes que el resto) de cualquier archivo de código fuente.  
`global using System.Linq;`

© JMA 2016. All rights reserved

# Modificadores de Miembros

Modificador	Tipo	Descripción
static	Estático o de clase	Pertenece al propio tipo (la clase) no a las instancias. Se puede utilizar con campos, métodos, propiedades, operadores, eventos y constructores, pero no con indizadores, destructores o tipos.
virtual	Virtual	Indica que puede reemplazarse en una clase derivada. Incompatible con static, override y abstract.
override	Sobrescribe	Reemplaza un método, propiedad, indizador o evento heredado. El elemento en la clase base debe estar marcado como virtual. Incompatible con new, static, virtual y abstract.
abstract	Abstracta	Marca al método, propiedad, indizador o evento como abstracto, no implementado (sin cuerpo). Son implícitamente virtual. Las clases derivadas están obligadas a redefinirlos salvo que sean abstractas. Incompatible con static.
unsafe	Inseguro	Denota un contexto no seguro, es necesario para cualquier operación que involucre a punteros.
extern	Externo	Indica que el método se implementa externamente, no implementado (sin cuerpo). Se usa normalmente con el atributo <code>[DllImport("????.dll")]</code> .

© JMA 2016. All rights reserved

# Constantes y Atributos

- Constantes

[attributes] [modifiers] const type declarators = value;

- Los modificadores permitidos son new y los cinco modificadores de acceso.

- Atributos o campos

[attributes] [modifiers] <tipoVariable> <nombreVariable> =  
 <valorInicial>, ...;

- Los modificadores permitidos son new, static, readonly, volatile y los cinco modificadores de acceso.
- El modificador volatile indica que un campo puede ser modificado en el programa por el sistema operativo, el hardware o un subproceso en ejecución de forma simultánea.
- Las asignaciones a los campos readonly sólo pueden tener lugar en la propia declaración o en un constructor de la misma clase.

© JMA 2016. All rights reserved

# Procedimientos y funciones

<tipoRetorno> <nombreFunción>(<ListaDeParámetros>) { ... }

**void** <nombreProcedimiento>(<ListaDeParámetros>) { ... }

- Lista de parámetros

- Por valor: <tipo> <Nombre>
- Por referencia: **ref** <tipo> <Nombre>, debe estar instanciado.
  - De entrada: **in** <tipo> <Nombre> (ver. 7.2) equivale a **ref readonly** (ver. 12)
  - De salida: **out** <tipo> <Nombre>
- Número variable: **params** <tipo>[] <Nombre>, de 0 a n. (Último de la firma)
- Opcionales (ver. 4.0): <tipo> <Nombre> = <ValorPorDefecto> (Últimos de la firma)

- Devolución de valores: **return** [expression];

- Invocación

<nombreFunción>(<const>, <var>, **ref** <var>, **out** <var>)

- Argumentos con nombre (ver. 4.0):
 

<nombreFunción>(<Nombre>; <const> o <var>, ...)
- Saltar argumentos opcionales:
 

<nombreFunción>(<const>, <NombreArg>, <var>)

- Firma del método: número, orden y tipos de los parámetros

© JMA 2016. All rights reserved

# Métodos

- Declaración en la interfaz:  
`[atributes] [modifiers] <tipoRetorno> [<interfaz>.]<nombreFunción>(<ListaDeParámetros>) { ... }`  
`[atributes] [modifiers] void [<interfaz>.]<nombreProcedimiento>(<ListaDeParámetros>) { ... }`  
`[atributes] [modifiers] <tipoRetorno> [<interfaz>.]<nombreFunción>(<ListaDeParámetros>);`  
`[atributes] [modifiers] void [<interfaz>.]<nombreProcedimiento>(<ListaDeParámetros>);`
- Los modificadores permitidos son new, static, virtual, sealed, override, abstract, extern y los cinco modificadores de acceso.
- Declaración en la interfaz:  
`[atributes] [new] <tipoRetorno> [<interfaz>.]<nombreFunción>(<ListaDeParámetros>);`  
`[atributes] [new] void [<interfaz>.]<nombreProcedimiento>(<ListaDeParámetros>);`
- Métodos parciales (ver. 3.0)
  - Se pueden definir en una parte de una declaración de tipo e implementarse en otra.
  - La implementación es opcional; si ninguna parte implementa el método parcial, la declaración de método parcial y todas sus llamadas se quitan de la declaración de tipo resultante de la combinación de las partes.

```
partial void <nombreProcedimiento>(<ListaDeParámetros>); // Declaración  
partial void <nombreProcedimiento>(<ListaDeParámetros>) { ... } // Implementación
```
- Métodos automáticos con cuerpo de expresión (ver. 6.0)  
`<tipoRetorno> [<interfaz>.]<nombreFunción>(<ListaDeParámetros>) => expresión;`

© JMA 2016. All rights reserved

# Métodos asincrónicos (ver. 5.0)

- Las palabras clave **async** y **await** permiten crear un método asincrónico casi tan fácilmente como se crea un método sincrónico.

```
async string MetodoAsync() {  
    Task<string> tarea = ...;  
    // ...  
    string rslt = await tarea;  
    // ...  
    return rslt;  
}
```

© JMA 2016. All rights reserved

# Auto referencia y Ámbito

- **this**

- Clases

- Referencia a la instancia de la clase que se está implementando.
- Operador de ámbito que resuelve los conflictos de nombre entre parámetros y atributos.

- Estructuras

- Operador de ámbito que resuelve los conflictos de nombre entre parámetros y atributos.

- **base**

- Clases

- Operador de ámbito que permite el acceso a la parte heredada.
- Solo permite a la clase base inmediata.

© JMA 2016. All rights reserved

# Constructores y destructores.

- Constructores:

```
[attributes] [modifiers] <NombreClase>([formal-parameter-list]) [<Inicializador>] { ... }
[attributes] static <NombreClase>() { ... } // Constructor de clase
– Los constructores de instancia no se heredan.
– Los constructores se pueden sobrecargar.
– Los constructores no se pueden invocar directamente, salvo con el operador new o los inicializadores. Si se marcan como privados impiden la instanciación desde fuera de la clase.
– Si la clase no tiene constructor, se genera automáticamente un constructor predeterminado público sin parámetros y los campos del objeto se inicializan con los valores predeterminados.
– Los constructores principales ya no están restringidos a los tipos record (ver: 12):
    public class Widget(string name, int width, int height, int depth) : NamedItem(name) {
– Se puede realizar la inicialización de objetos sin llamadas explícitas a un constructor: (ver. 3.0)
    Clase C = new Clase { Propiedad1=v1, P2=v2 };

```

- Finalizadores (anteriormente conocidos como destructores):

```
[attributes] ~<NombreClase> () { ... }
– Los destructores no se pueden heredar ni sobrecargar.
– No se puede llamar a los destructores. Se invocan automáticamente.
```

- Inicializador:

```
– : base (argument-list)
– : this (argument-list)
```

- Los modificadores permitidos son extern y los cinco modificadores de acceso.

© JMA 2016. All rights reserved

# Sobrecarga de Operadores

---

```
public static result-type operator unary-operator (this-type operand)
public static result-type operator binary-operator (this-type operand1,
    op-type operand2)
public static result-type operator binary-operator (op-type operand1,
    this-type operand2)
public static result-type operator binary-operator (op-type op, int
    despla) // << >>
public static implicit operator conv-type-out ( conv-type-in operand )
public static explicit operator conv-type-out ( conv-type-in operand )
```

- Operadores unarios sobrecargables:  
+ - ! ~ ++ -- true false
- Operadores binarios sobrecargables:  
- + - \* / % & | ^ << >> == != > < >= <=

---

© JMA 2016. All rights reserved

# Delegados

- 
- Define un tipo de referencia que se puede utilizar para encapsular un método con una firma específica (similar a un puntero a función con tipo).  
[attributes] [modifiers] **delegate** result-type identifier ([formal-parameters]);
  - Los modificadores permitidos son new y los cinco modificadores de acceso.  
 $\langle \text{TipoDelegado} \rangle \langle \text{Variable} \rangle = \text{new}$   
 $\langle \text{TipoDelegado} \rangle (\langle \text{Objeto} \rangle . \langle \text{Método} \rangle); // \text{Antes del 2.0}$   
 $\langle \text{TipoDelegado} \rangle \langle \text{Variable} \rangle = \langle \text{Objeto} \rangle . \langle \text{Método} \rangle;$
  - Donde la firma de  $\langle \text{Objeto} \rangle . \langle \text{Método} \rangle$  debe coincidir con la declarada en  $\langle \text{TipoDelegado} \rangle$ .
  - Métodos anónimos:  
**delegate**([formal-parameters]) { ... Cuerpo... }

---

© JMA 2016. All rights reserved

# Tipos dinámicos, Anónimos y Expresiones lambda

- Uso de tipo dinámico (ver. 4.0)
 

```
dynamic variable;
```

  - Se trata de un tipo estático, pero un objeto de tipo dynamic omite la comprobación de tipos en tiempo de compilación.
  - En la mayoría de los casos, funciona como si tuviera el tipo object.
  - En tiempo de compilación, se supone que un elemento de tipo dynamic admite cualquier operación.
- Tipos anónimos (ver. 3.0)
 

```
var x = new { prop1 = val1, prop2 = val2, ... }
```

  - Habilita la creación inmediata de tipos estructurados sin nombre que se pueden agregar a colecciones y a los que se puede tener acceso utilizando var.
- Expresiones lambda (ver. 3.0)
 

```
(input parameters) => expression
```

  - Habilita expresiones insertadas con parámetros de entrada que se pueden enlazar a delegados o árboles de expresión.
  - Son funciones anónimas y simplifican su creación. Equivale a:
 

```
delegate(input parameters) { return expresión; }
```
  - Los parámetros toman el tipo de la definición en la invocación. Los paréntesis son opcionales cuando el parámetro es único. Se puede proporcionar valores predeterminados para los parámetros (ver. 12).
  - Si el cuerpo de la función requiere algo más que una expresión, se crea un bloque que requiere un return explícito.

© JMA 2016. All rights reserved

# Propiedades

- [attributes] [modifiers] type [<interfaz>.]identifier {< Descriptores de acceso>}
- Los modificadores permitidos son new, static, virtual, sealed, override, abstract, extern y los cinco modificadores de acceso.
  - Descriptores de acceso
 

```
[modifiers] get {...}
```

```
[modifiers] set { ... value ... }
```

    - Los modificadores permitidos son los cinco modificadores de acceso.
  - Declaración en la interfaz:
 

```
[attributes] [new] type identifier { get; set; }
```
  - Propiedades auto implementadas (**ver. 3.0**)
 

```
[attributes] [modifiers] type identifier { get; set; }
```
  - Propiedades auto implementadas solo lectura (**ver. 6.0**)
 

```
[attributes] [modifiers] type identifier {get; private set;}
```

```
[attributes] [modifiers] type identifier {get; }
```
  - Inicialización de las propiedades auto implementadas (**ver. 6.0**)
 

```
[attributes] [modifiers] type identifier {get; set;} = value;
```

© JMA 2016. All rights reserved

# Propiedades

- Establecedores de solo inicialización.
  - A partir de C# 9.0, puede crear descriptores de acceso init en lugar de descriptores de acceso set para propiedades e indizadores de solo lectura.
- Inicialización de instancias:
  - Además de con el constructor, se pueden inicializar con propiedades:

```
var now = new WeatherObservation {
    RecordedAt = DateTime.Now,
    TemperatureInCelsius = 20,
    PressureInMillibars = 998.0m
};
```

© JMA 2016. All rights reserved

# Indizadores

- 
- [attributes] [modifiers] type <interfaz>.this [formal-index-parameter-list]  
 {<DescriptoresDeAcceso>}
- [attributes] [new] type this [formal-index-parameter-list] {get;set;}
- Los modificadores permitidos son new, virtual, sealed, override, abstract, extern y una combinación válida de los cinco modificadores de acceso.
  - Descriptores de acceso
 

```
get {...}
set { ... value ... }
```
  - Para proporcionar al indizador un nombre que puedan utilizar otros lenguajes para la propiedad indizada predeterminada, se debe utilizar el atributo:
 

```
[System.Runtime.CompilerServices.CSharp.IndexerName("<Nombre>")]
```

    - El indizador tendrá el nombre <Nombre>. Si no se especifica el atributo de nombre, el nombre predeterminado será Item.
  - Declaración en la interfaz:
 

```
[attributes] [new] type this [formal-index-parameter-list] {get;set;}
```
  - **Inicializador del indizador (*ver. 6.0*)**

```
new Colección() { [índice] = valor, ... }
```

© JMA 2016. All rights reserved

# Iteradores

- Métodos que calculan y devuelven una secuencia de valores
- Debe devolver `IEnumerator` o `IEnumerable`
- La sentencia `foreach` se apoya en un patrón de enumeración
- Las interfaces del enumerador son `System.Collections.IEnumerator` y los tipos construidos a partir de `System.Collections.Generic.IEnumerator<T>`.
- Las interfaces enumerables son `System.Collections.IEnumerable` y los tipos construidos a partir de `System.Collections.Generic.IEnumerable<T>`.
- Instrucciones:
  - `yield return <valor>`; genera el siguiente valor de la iteración
  - `yield break`; indica que se completó la iteración

```
public IEnumerator<T> GetEnumerator() {
    for (int i = count - 1; i >= 0; --i) {
        yield return items[i];
    }
}
```

© JMA 2016. All rights reserved.

# Tipos genéricos

- Pueden ser clases, estructuras, interfaces, delegados y métodos.
- La definición se realiza mediante alias de posibles tipos.
- En el momento de su utilización se resuelven los alias con tipos existentes.
- Se pueden restringir los tipos posibles de sustitución de alias.
- Declaraciones de clases genéricas
 

```
[atributes] [modifiers] class nombre<listaAliasDeTipo> [:base-list] [lista de restricciones de tipo]{  
    ... Miembros ...  
}
```
- Declaraciones de estructuras genéricas
 

```
[atributes] [modifiers] struct nombre<listaAliasDeTipo> [:ListaDeInterfces] [lista de  
restricciones de tipo]{  
    ... Miembros ...  
}
```
- Declaraciones de Interfaces genéricas
 

```
[atributes] [modifiers] interface nombre<listaAliasDeTipo> [:ListaDeInterfces] [lista de  
restricciones de tipo]{  
    ... Miembros ...  
}
```

© JMA 2016. All rights reserved.

# Tipos genéricos

- Declaraciones de delegados genéricas  
`[attributes] [modifiers] delegate result-type  
    identifier<listaAliasDeTipo> ([formal-parameters]) [lista de  
    restricciones de tipo];`
- Declaraciones de métodos genéricos  
`[attributes] [modifiers] <tipoRetorno>  
    <nombreMétodo><listaAliasDeTipo> (<ListaDeParámetros>)  
    [lista de restricciones de tipo] { ... }`
- Restricción de tipo:
  - **where** AliasDeTipo: ... restricciones ...
  - Posibles restricciones (definición opcional, separadas por comas):
    - tipoClase, **class** ó **struct** (principio de la lista)
    - lista de interfaces
    - Otro alias de la listaAliasDeTipo
  - **new()** (al final de la lista)

© JMA 2016. All rights reserved

# Eventos

---

`[attributes] [modifiers] event type declarator;`  
`[attributes] [modifiers] event type member-name {accessor-declarations};`

- Descriptores de acceso  
`add { ... value ... }`  
`remove { ... value ... }`
- Declaración en la interfaz:  
`[attributes] [new] event type declarator;`
- Patrón estándar:  
`void <Evento>(object sender, <HerederoDeEventArgs> e)`
- Conceptos:
  - Cualquier objeto capaz de producir un evento es un remitente de eventos.
  - Cualquier objeto capaz de contener un remitente de eventos, puede ser un consumidor de eventos.
  - Los controladores de eventos son procedimientos llamados cuando se produce un evento correspondiente.

© JMA 2016. All rights reserved

# Eventos

Pasos a seguir:

1. En caso de ser necesario, definir una clase que proporcione los datos del evento. Esta clase debe derivar de System.EventArgs, que es la clase base de los datos del evento.
2. En caso de ser necesario, declarar un tipo delegado para el evento (Action<>).
3. Obligatoriamente, definir un miembro de evento público en la clase remitente de eventos.
4. Opcionalmente, incluir en la clase remitente un método protegido que provoque el evento. Este método se debe denominar OnNombreEvento. El método OnNombreEvento provoca el evento invocando a los delegados. Antes de provocar el evento es necesario comprobar que el evento está asociado a controladores de eventos, en caso contrario, si no se encuentra asociado generará una excepción.  
`if(<NombreEvento> != null) <NombreEvento>(this, e);`
5. En los métodos apropiados provocar el evento o invocar, en caso de que este incluido, al método que lo provoca.
6. Crear los controladores de eventos, normalmente en la clase consumidora de eventos aunque no obligatoriamente. Debe tener la misma firma que el delegado del evento.
7. Asociar o registrar, en el consumidor de eventos, el controlador de eventos al evento de una instancia del remitente de eventos. Un controlador de eventos puede estar asociado a varios eventos de varias instancias siempre y cuando tengan la misma firma. Un evento puede tener asociados varios controladores de eventos (se ejecutan en el mismo orden en el que se han asociado).  
`<Instancia> <Evento> += new <TipoDelegadoEvento>(<Objeto>.<MétodoControlador>)`

Para desasociar un evento:

```
<Instancia> <Evento> -= new <TipoDelegadoEvento>(<Objeto>.<MétodoControlador>)
```

© JMA 2016. All rights reserved

## Extensiones para el código no seguro

- Marcar:  
`externopt unsafeopt static  
unsafeopt externopt static  
externopt static unsafeopt  
unsafeopt static externopt  
static externopt unsafeopt  
static unsafeopt externopt  
unsafe block`
- Punteros:  
`Type* p = &var;`
- La instrucción **fixed**, que se utiliza para “fijar” una variable móvil de manera que su dirección permanece constante en toda la duración de la instrucción:  
`fixed ( pointer-type fixed-pointer-declarators ) embedded-statement`
- El operador sizeof devuelve el número de bytes ocupados por una variable de un tipo dado:  
`sizeof ( unmanaged-type )`
- Una declaración de variable local puede incluir un inicializador de asignación de pila que asigne memoria de la pila de llamadas:  
`stackalloc unmanaged-type [ expression ]`

© JMA 2016. All rights reserved

## Clases estáticas

- Las clases estáticas formalizan el patrón de diseño de la “clase sellada no instanciable”.
- Todos sus miembros deben ser estáticos.
- Son clases de utilidad.  
`[attributes] [modifiers] static class nombre [:base-list] {  
... Miembros estáticos o de clase ...  
};}`
- Métodos de extensión (**ver. 3.0**)
 

```
public static <tipoRetorno> <nombreMétodo>(this <tipoClaseAExtender>  
<ParmConRefAExtender>, <ListaDeParámetros>) { ... }
```

  - Deben estar declarados en clases estáticas (clases con el sufijo Extensions).
  - Extienden las clases existentes con métodos estáticos que puedan invocarse mediante la sintaxis de método de instancia de las clases existentes.
  - Para poder ser usados se debe importar explícitamente (using) su espacio de nombres.

© JMA 2016. All rights reserved

## Expresiones de consulta (ver. 3.0)

- Aparecen para dar soporte a Linq
- Palabras clave que especifican cláusulas en una expresión de consulta:
  - Cláusulas from
  - Cláusula where (opcional)
  - Cláusulas de ordenación (opcional)
  - Cláusula join (opcional)
  - Cláusula select o group
  - Cláusula into (opcional)

```
from typeopt identifier in expression
let identifier = expression
where boolean-expression
join typeopt identifier in expression on expression equals expression
join typeopt identifier in expression on expression equals expression into identifier
orderby orderings
expression ordering-ascendingopt descendingopt
select expression
group expression by expression
into identifier query-body
```

- Ej: from ... into x ... from x in (from ... ) ...

© JMA 2016. All rights reserved

## REFLEXIÓN

© JMA 2016. All rights reserved

### Reflexión

- La clase Assembly representa un ensamblado, que es un bloque de compilación reutilizable, versionable y autodescriptivo.
- Los métodos Load permiten cargar dinámicamente ensamblados.
- Los métodos GetFiles y GetFile permiten acceder a los ficheros almacenados en el área de recursos del ensamblado.
- Los métodos GetTypes y GetType permiten obtener las clases de un ensamblado.
- El método CreateInstance crea una instancia de una clase partiendo de una cadena con el nombre de la clase (instanciación dinámica)

© JMA 2016. All rights reserved

## Reflexión

- La clase `Type` representa la clase de una clase, incluye métodos que retornan información sobre la clase:
  - `GetMembers()`, `GetFields()`, `GetMethods()`,  
`GetConstructors()`, `GetProperties()`, `GetEvents()`,  
`GetInterfaces()`, ...
- `FieldInfo`, `MethodInfo` y `ConstructorInfo`,  `PropertyInfo` y `EventInfo` representan la información sobre atributos, métodos, propiedades y eventos.
- Obtención del tipo:
  - `Type t = instancia.GetType();`
  - `Type t = typeof(clase);`

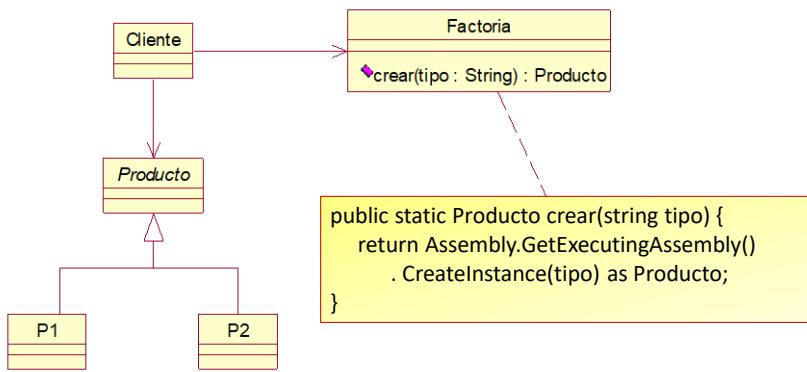
© JMA 2016. All rights reserved

## Reflexión

- Se puede utilizar el `Type.InvokeMember` para invocar un miembro de un tipo.
- Utilizando una cadena con el nombre del miembro y una instancia de la clase representada por `Type` se puede:
  - Obtener el valor de un atributo o una propiedad de la instancia.
  - Modificar el valor de un atributo o una propiedad de la instancia.
  - Invocar un método de la instancia, pasándole parámetros y obteniendo el valor de retorno.
- También es posible utilizarlo para crear instancias invocando al constructor, pasándole parámetros en caso de ser necesario.

© JMA 2016. All rights reserved

# Factoría y Metaclases



© JMA 2016. All rights reserved.

## COLECCIONES

© JMA 2016. All rights reserved.

# Introducción

- A menudo, los datos similares pueden controlarse de forma más eficaz si se almacenan y manipulan como si fuesen una colección. Puede usar la clase System.Array pero presentan serias limitaciones al tener que estar dimensionados desde el principio. Las colecciones permiten agregar, quitar y modificar elementos individuales o intervalos de elementos de forma dinámica: ICollection o ICollection<T>. Las diferentes implementaciones suministran comportamientos adicionales al almacenamiento.
- Hay dos tipos principales de colecciones: las colecciones genéricas y las colecciones no genéricas. Las colecciones genéricas se agregaron en la versión 2.0 de .NET Framework, son colecciones fuertemente tipadas, seguridad de tipos en tiempo de compilación, y normalmente ofrecen un mejor rendimiento. Las colecciones genéricas aceptan un parámetro de tipo cuando se construyen y no requieren conversiones con el tipo Object al agregar o quitar elementos de la colección.
- A partir de .NET Framework 4, las colecciones del espacio de nombres System.Collections.Concurrent proporcionan operaciones eficaces y seguras para subprocesos con el fin de obtener acceso a los elementos de la colección desde varios subprocesos. Las clases de colección inmutables en el espacio de nombres System.Collections.Immutable (NuGet package) son intrínsecamente seguras para los subprocesos, ya que las operaciones se realizan en una copia de la colección original, mientras que la colección original no se puede modificar.

© JMA 2016. All rights reserved

## Array vs Collection vs Dictionary

### Array

- Necesidad de tamaño fijo
- Acceso aleatorio a elementos
- Tipo de datos fijos
- Array dinámicos necesitan de estructura de indización

### Collection

- Encapsulan funcionamiento interno
- Variedad de tipos adaptables
- Posibilidad de personalización de características
- Son iterables

### Dictionary

- Almacenan claves asociadas a valores
- Estructura dinámica como las colecciones
- Permiten buscar y recuperar más fácil y rápidamente que en la mayoría de las colecciones

© JMA 2016. All rights reserved

# Selección de una colección

- Con acceso a elementos por índice:
  - IList<T>, Array, ArrayList, List<T>, ImmutableList<T>, ImmutableArray
- Un conjunto (sin repetición):
  - ISet<T>, HashSet<T>, SortedSet<T>, ImmutableHashSet<T>, ImmutableSortedSet<T>
- Con estructura de cola FIFO (el primero en entrar es el primero en salir):
  - Queue, Queue<T>, ConcurrentQueue<T>, ImmutableQueue<T>
- Con estructura de pila LIFO (el último en entrar es el primero en salir):
  - Stack, Stack<T>, ConcurrentStack<T>, ImmutableStack<T>
- Para acceso a elementos de forma secuencial:
  - LinkedList<T>
- Con notificaciones cuando se quitan o se agregan elementos a la colección. (implementa INotifyPropertyChanged y INotifyCollectionChanged):
  - ObservableCollection<T>
- Una colección ordenada:
  - SortedList< TKey, TValue >, ImmutableSortedDictionary< TKey, TValue >, ImmutableSortedSet< T >
- Como alternativa a las colecciones, los diccionarios o mapas permiten almacenar elementos como pares clave/valor para una consulta rápida por clave:
  - IDictionary, Hashtable, Dictionary< TKey, TValue >, ConcurrentDictionary< TKey, TValue >, IReadOnlyDictionary< TKey, TValue >, ImmutableDictionary< TKey, TValue >

© JMA 2016. All rights reserved

## List<T>

- Propiedades
  - Count
  - Item [int]
- Métodos
  - Add(T elemento)
  - AddRange(Collection c)
  - Clear()
  - Contains(T elemento)
  - Find( <<predicado>> )
  - Insert(int, T)
  - Remove(T)
  - RemoveAt(int)
  - Sort()
  - TrueForAll(<<predicado>> )

© JMA 2016. All rights reserved

## List<T>

```
List<int> lista = new List<int>();  
lista.Add(10);  
lista.Add(20);  
lista.Add(30);  
foreach (var valor in lista)  
    Console.WriteLine(valor);  
var i = lista[1];  
lista.RemoveAt(1);  
bool pares = lista.TrueForAll(e => e%2 == 0);
```

© JMA 2016. All rights reserved

Language-Integrated Query

# INTRODUCCIÓN A LINQ

© JMA 2016. All rights reserved

## ¿Qué es LINQ?

- Mecanismo uniforme y extensible para consultar fuentes de datos de diferentes tipos: las expresiones de consulta.
- Sintaxis basada en nuevas palabras reservadas contextuales.
- Semántica “enchufable”: los lenguajes no definen la semántica de las nuevas palabras reservadas, sino únicamente un conjunto de reglas para reescribir esas expresiones como cascadas de llamadas a métodos.

© JMA 2016. All rights reserved

## Características LINQ

- Sintaxis familiar para escribir consultas (“Parecida” a SQL).
- Sintaxis unificada independientemente de la fuente de datos.
- Esfuerzo enfocado en el negocio y no en el acceso a datos.
- Comprobación en tiempo de compilación de errores de sintaxis y seguridad de tipos.
- Compatibilidad con IntelliSense.
- Eficaces funcionalidades de filtrado, ordenación y agrupación.
- Simplicidad de código y orientado a objetos.
- Transaccional.
- Basado en Lambda Cálculo, Tipado fuerte, Ejecución retrasada (deferred), Inferencia de tipos, Tipos anónimos, Métodos extensores y Inicialización de objetos

© JMA 2016. All rights reserved

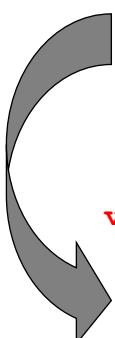
## Lista de providers

- LINQ to SQL
- LINQ to XML
- LINQ to Objects
- LINQ to DataSets
- LINQ To Active Directory
- LINQ To Amazon
- LINQ To mysql, Oracle, PostgreSQL
- LINQ To JSON
- LINQ To Sharepoint
- LINQ To JavaScript
- LINQ To Excel

© JMA 2016. All rights reserved

## Expresiones de consulta

```
var delMadrid =  
    from f in DatosFutbol.Futbolistas  
    where f.CodigoClub == "RMA"  
    select new { f.Nombre, f.Edad };
```



```
var delMadrid =  
    DatosFutbol.Futbolistas  
    .Where(f => f.CodigoClub == "RMA")  
    .Select(f => new { f.Nombre, f.Edad });
```

© JMA 2016. All rights reserved

# Expresión de Consulta

```
from id in source
{from id in source |
 join id in source on expr equals expr [ into id ] |
 let id = expr |
 where condition |
 orderby ordering, ordering, ... }
select expr | group expr by key
[ into id query ]
```

Empieza con  
*from*

Cero o más *from, join,*  
*let, where, o orderby*

Termina con  
*select o group by*

Continuación *into*  
opcional

© JMA 2016. All rights reserved.

## Expresiones de consulta

- Fuentes de consultas
  - Los datos provienen de cierta fuente, que implementa `IEnumerable<T>`.
- Operadores de consulta estándar
  - No todos los operadores tienen un reflejo en la sintaxis de los lenguajes.
  - El patrón LINQ.

© JMA 2016. All rights reserved.

# Operadores

Restricción	Where
Proyección	Select, SelectMany
Ordenación	OrderBy, ThenBy
Agrupación	GroupBy
Encuentros	Join, GroupJoin
Cuantificadores	Any, All
Partición	Take, Skip, TakeWhile, SkipWhile
Conjuntuales	Distinct, Union, Intersect, Except
Un elemento	First, Last, Single, ElementAt
Agregados	Count, Sum, Min, Max, Average
Conversión	ToArray,ToList, ToDictionary
Conversión de elementos	OfType<T>, Cast<T>

© JMA 2016. All rights reserved

# Operadores de Consulta

Expresión de consulta de Linq	Where(), Select(), SelectMany(), OrderBy(), ThenBy(), OrderByDescending(), ThenByDescending(), GroupBy(), Join(), GroupJoin()
Partición	Take(), Skip(), TakeWhile(), SkipWhile()
Conjunto	Distinct(), Union(), Intersect(), Except()
Conversión	ToArray(),ToList(), ToDictionary(), ToLookup(), AsEnumerable(), Cast<T>(), OfType<T>()
Generación	Range(), Repeat<T>(), Empty<T>(), Concat(), Reverse()
Cuantificación	Any(), All(), Contains(), SequenceEqual()
Elementos	First(), Last(), Single(), ElementAt(), DefaultIfEmpty(). {método}OrDefault()
Agregados	Count(), LongCount(), Max(), Min(), Sum(), Average(), Aggregate()

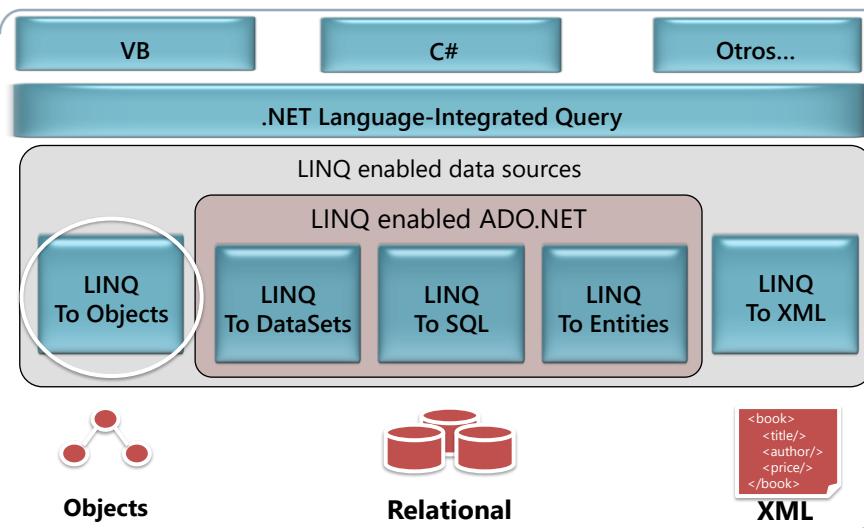
© JMA 2016. All rights reserved

# Expresiones de consulta

- Composicionales, jerárquicas
  - Anidamiento arbitrario.
  - Posibilidad de aplicar operadores adicionales.
- Declarativas y no imperativas
  - Diga qué se desea obtener, no cómo.
  - El cómo va por el proveedor.
- Ejecución diferida
  - Las consultas se ejecutan solo a medida que sus resultados se solicitan.

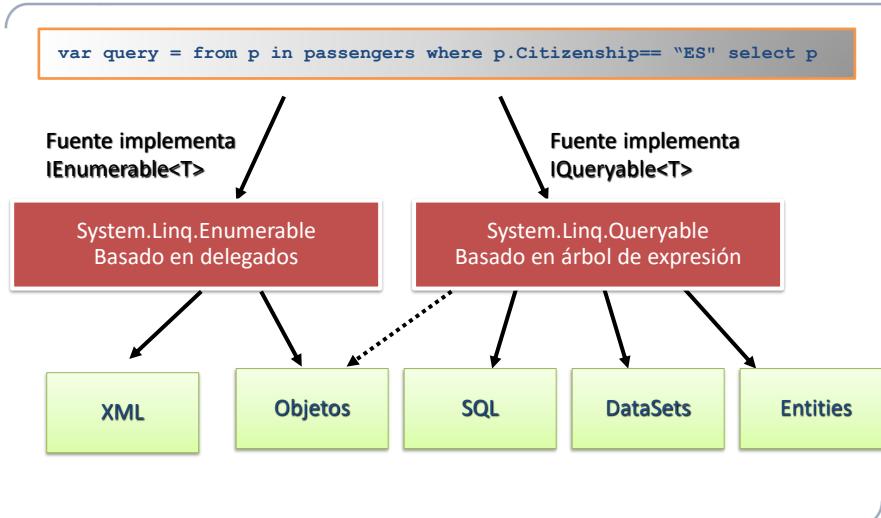
© JMA 2016. All rights reserved

## Language INtegrated Query (LINQ)



© JMA 2016. All rights reserved

# Arquitectura de LINQ



© JMA 2016. All rights reserved.

## Dos clases de proveedores

	Basados en <code>IEnumerable&lt;T&gt;</code>	Basados en <code>IQueryable&lt;T&gt;</code>
Interfaz	<code>IEnumerable&lt;T&gt;</code>	<code>IQueryable&lt;T&gt;</code>
Ejecución	Local, en memoria	Usualmente remota
Implementación	Iteradores	Ánálisis de árboles de expresiones
Proveedores	<code>LINQ to Objects</code> <code>LINQ to XML</code> <code>LINQ to DataSet</code>	<code>LINQ to SQL</code> <code>LINQ to Entities</code>
Más ejemplos	<code>LINQ to Pipes</code> <code>LoggingLINQ</code>	<code>LINQ to TFS</code>

© JMA 2016. All rights reserved.

## Extendiendo LINQ

- Habilite sus API existentes para LINQ
  - Específicamente para consultas en memoria.
  - Cree métodos extensores que devuelvan un objeto `IEnumerable<T>`.
- Desarrolle su propio proveedor de consultas
  - Implemente `IQueryable<T>`.
  - Analice árboles de expresiones y traduzca nodos a código o a un lenguaje de consultas diferente.

© JMA 2016. All rights reserved

## Recomendaciones

- Analice cuándo y cómo sus consultas se ejecutan
  - Momento de ejecución.
  - Ejecución local vs. remota.
  - Lugar/capa de ejecución real.
- Mantenga las consultas dentro de ensamblados
  - No pase expresiones de consulta entre capas.

© JMA 2016. All rights reserved

## Recomendaciones (2)

- Cuidado con los tipos anónimos!
  - Planifique de antemano qué tipos son importantes.
  - No abuse de las proyecciones.
- Aprenda:
  - A escribir consultas con y sin la sintaxis.
  - Las nuevas características de C# 3.0
  - Los detalles de la traducción de la sintaxis en llamadas a operadores y cómo funcionan éstos.

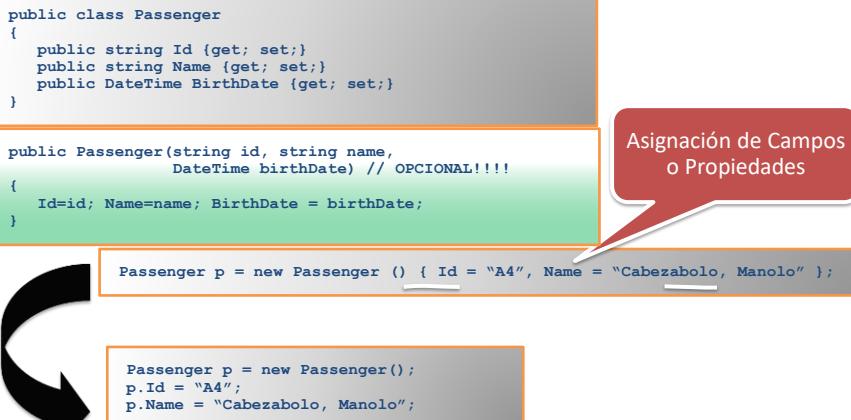
© JMA 2016. All rights reserved

## Nuevas características

- Inicializadores de objetos
- Inferencia de tipos
- Tipos anónimos
- Métodos extensores
- Expresiones lambda
- Árboles de expresión
- LINQ!!!

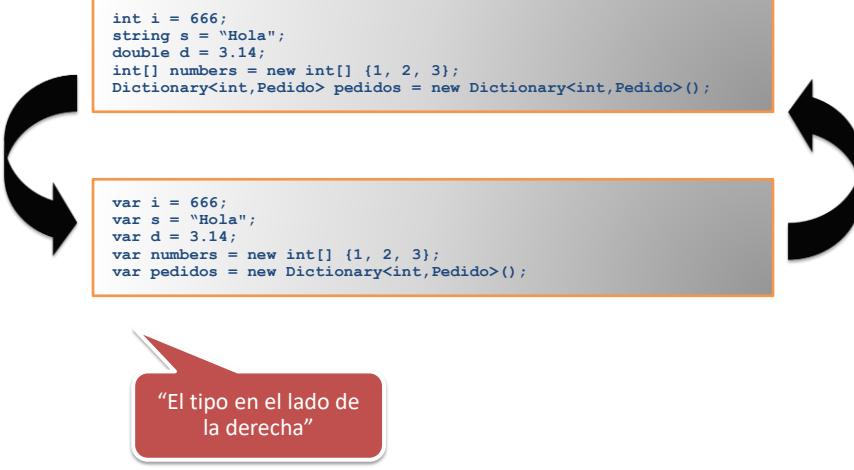
© JMA 2016. All rights reserved

# Inicializadores de Objetos



© JMA 2016. All rights reserved

# Inferencia de Tipos



© JMA 2016. All rights reserved

# Tipos Anónimos

XXX

```
class XXX
{
    public string Name;
    public int Age;
}
```

```
var o = new { Name = "Pantoja", Age= 75 };
```

© JMA 2016. All rights reserved

## Métodos Extensos

Método extensor

```
namespace MisCosas
{
    public static class Extensiones
    {
        public static string Concatenar(this IEnumerable<string> strings,
                                       string separador) {...}
    }
}
```

using MisCosas;

Incluir extensiones en el ámbito

```
string[] nombres = new string[] { "Edu", "Juan", "Manolo" };
string s = nombres.Concatenar(", ");
```

IntelliSense!

obj.Foo(x, y)  
↓  
XXX.Foo(obj, x, y)

© JMA 2016. All rights reserved

# Expresiones Lambda

```

public delegate bool Predicate<T>(T obj);

public class List<T>
{
    public List<T> FindAll(Predicate<T> test) {
        List<T> result = new List<T>();
        foreach (T item in this)
            if (test(item)) result.Add(item);
        return result;
    }
    ...
}

```

Delegado genérico

Tipo genérico

© JMA 2016. All rights reserved

# Expresiones Lambda

```

public class MiClase
{
    public static void Main() {
        List<Cliente> clientes = ObtenerListaClientes();
        List<Cliente> locales =
            clientes.FindAll(
                new Predicate<Cliente>(CiudadIgualCoruna)
            );
    }

    static bool CiudadIgualCoruna(Cliente c) {
        return c.Ciudad == "A Coruña";
    }
}

```

© JMA 2016. All rights reserved

# Expresiones Lambda

```
public class MiClase
{
    public static void Main() {
        List<Cliente> clientes = ObtenerListaClientes ();
        List<Cliente> locales =
            clientes.FindAll(
                delegate(Cliente c) { return c.Ciudad == "A Coruña"; }
            );
    }
}
```

Delegado  
Anónimo

© JMA 2016. All rights reserved

# Expresiones Lambda

```
public class MiClase
{
    public static void Main() {
        List<Cliente> clientes = ObtenerListaClientes ();
        List<Cliente> locales =
            clientes.FindAll(
                (Clientes c) => {return c.Ciudad == "A Coruña"; }
            );
    }
}
```

Expresión  
Lambda

© JMA 2016. All rights reserved

## Expresiones Lambda

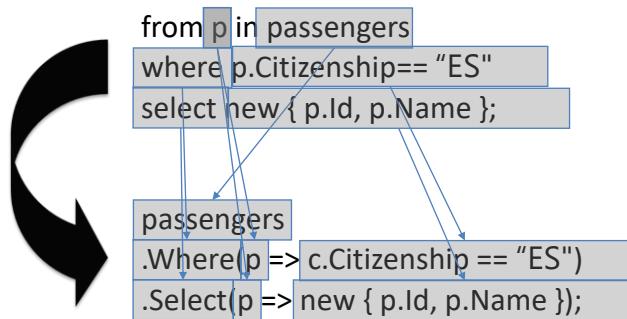
```
public class MiClase
{
    public static void Main() {
        List<Cliente> clientes = ObtenerListaClientes ();
        List<Cliente> locales =
            clientes.FindAll(c => c.Ciudad == "A Coruña");
    }
}
```

Expresión  
Lambda

© JMA 2016. All rights reserved

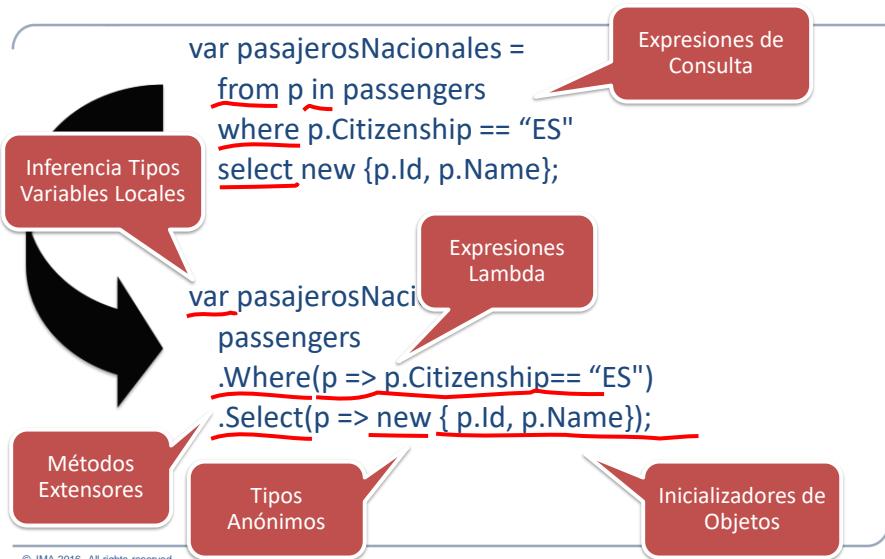
## Introduciendo LINQ

- Todos estos nuevos aspectos se trasladan a métodos extensores sobre colecciones:
  - Pueden transformarse en árboles de expresiones



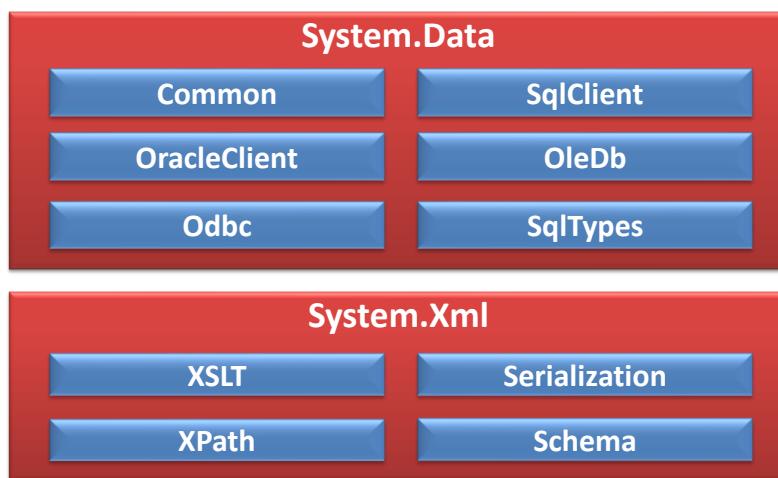
© JMA 2016. All rights reserved

## Introduciendo LINQ



## ADO.NET

## Acceso a Datos: ADO.NET



© JMA 2016. All rights reserved.

## Escenario Conectado

- Un entorno conectado es uno en el cual los usuarios están constantemente conectados a la fuente de datos
- Ventajas:
  - Mayor seguridad
  - Mejor control de concurrencia
  - Los datos se mantienen actualizados
- Desventajas:
  - Se requiere una conexión constante (consume recursos del servidor)
  - Escalabilidad

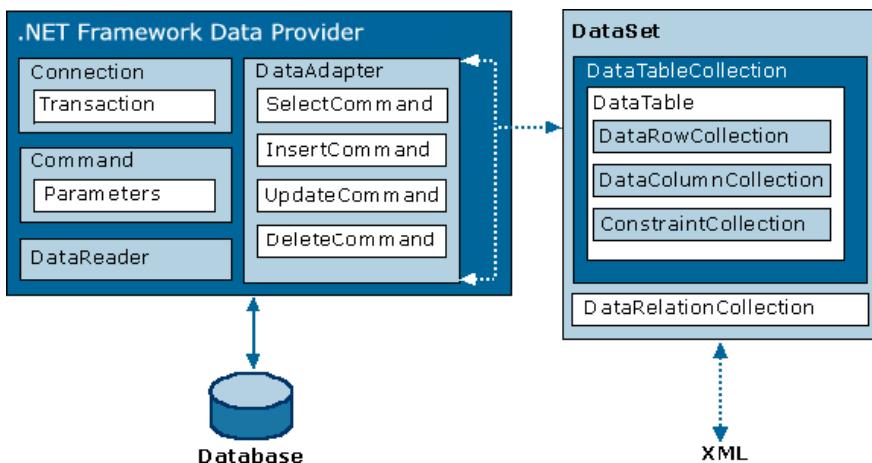
© JMA 2016. All rights reserved.

## Escenario Desconectado

- En un entorno desconectado, una parte de los datos del repositorio central se copia y modifica en forma local, para luego sincronizarse con éste.
- Ventajas
  - Se puede trabajar en forma independiente
  - Mayor escalabilidad y performance
- Desventajas
  - Los datos no están sincronizados
  - Resolución manual de conflictos

© JMA 2016. All rights reserved

## ADO.NET - Arquitectura



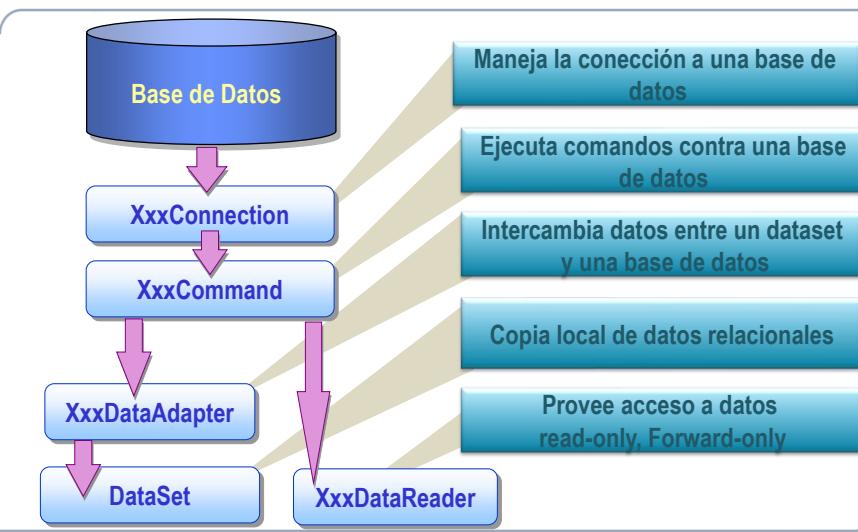
© JMA 2016. All rights reserved

## Proveedores de Acceso a Datos

- OLE DB (`System.Data.OleDb`)
- ODBC (`System.Data.Odbc`)
- SQL Server (`System.Data.SqlClient`)
- Oracle (`System.Data.OracleClient`)
- Otros provistos por terceros (MySQL, PostgreSQL, DB2, etc..)
- EntityClient (`System.Data.EntityClient`) para Entity Data Model (EDM)

© JMA 2016. All rights reserved

## Clases más comunes



© JMA 2016. All rights reserved

## Clases y miembros

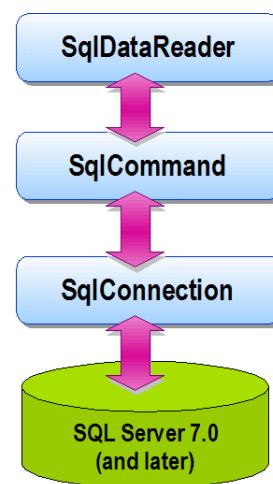
- **Connection:** Establece una conexión a un origen de datos determinado.
  - ConnectionString
  - Open, close, CreateCommand, BeginTransaction
- **Transaction:** Permite incluir comandos en las transacciones que se realizan en el origen de datos.
  - Commit, Rollback
- **Command:** Ejecuta un comando en un origen de datos.
  - Connection, Transaction, CommandType (Text, StoredProcedure, TableDirect), CommandText, CommandTimeout,
  - ExecuteReader, ExecuteNonQuery, ExecuteScalar, ExecuteXmlReader
  - UpdatedRowSource
  - Parameters
- **DataReader:** Lee una secuencia de datos de sólo avance y sólo lectura desde un origen de datos.
  - Item, GetXXXX, IsDBNull
  - Read, NextResult, Close

© JMA 2016. All rights reserved

## Accediendo a datos: Conectado

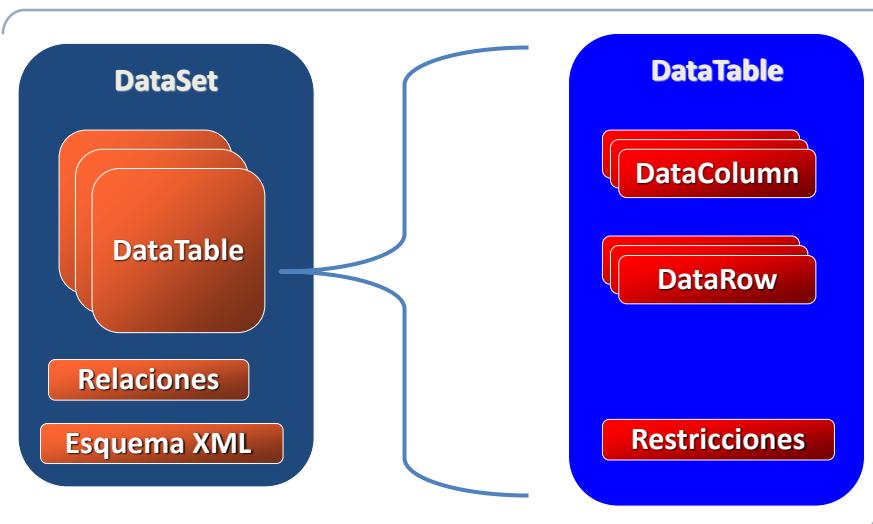
En un escenario conectado, los recursos se mantienen en el servidor hasta que la conexión se cierra:

1. Abrir Conexión
2. Ejecutar Comando
3. Procesar Filas en DataReader
  - Mientras Leer es verdadero
    - Procesar datos leídos
4. Cerrar DataReader
5. Cerrar Conexión



© JMA 2016. All rights reserved

# DataSet



© JMA 2016. All rights reserved.

## Clases y miembros

- **DataSet:** Representa una caché de datos en memoria.
  - Es independiente de la fuente de datos.
  - Puede leer y escribir datos y esquemas como documentos XML.
  - Puede migrar entre las capas de aplicación.
  - Puede imponer la integridad de los datos y permite navegar en la jerarquía de la tabla.
  - Con el Diseñador de DataSet se pueden crear conjuntos de datos con establecimiento inflexible de tipos mediante herencia y personalizables a través de partial class.
  - Miembros:
    - Tables, Relations, EnforceConstraints
    - HasChanges, GetChanges, AcceptChanges, RejectChanges
    - Clear, Merge
    - GetXml, ReadXml, WriteXml, WriteXmlSchema

© JMA 2016. All rights reserved.

# Clases y miembros

- **DataTable:** Representa una tabla de datos en memoria.
  - Columns, Constraints, ChildRelations, ParentRelations, DefaultView, Rows
  - NewRow, ImportRow, Select, Compute, HasErrors, GetErrors
  - ReadXml, WriteXml, WriteXmlSchema
  - HasChanges, GetChanges, AcceptChanges, RejectChanges, Clear, Merge
  - RowChanged, RowChanging, RowDeleting y RowDeleted, ColumnChanged, ColumnChanging
- **DataRow:** Representa una fila de datos en un DataTable.
  - Item, IsNull, SetNull, BeginEdit, EndEdit, CancelEdit, AcceptChanges, RejectChanges
  - Remove, HasVersion, RowState
  - GetParentRow, SetParentRow, GetChildRows
  - HasErrors, RowError, GetColumnError, GetColumnsInError, SetColumnError, ClearErrors

© JMA 2016. All rights reserved

# Clases y miembros

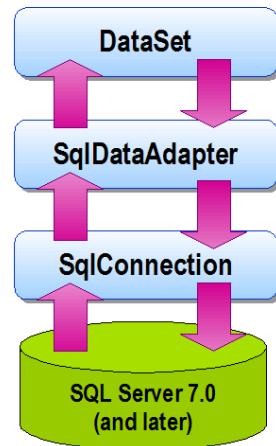
- **DataAdapter:** Llena un DataSet y realiza las actualizaciones necesarias en el origen de datos.
  - SelectCommand, InsertCommand, UpdateCommand y DeleteCommand
  - TableMappings, MissingMappingAction, MissingSchemaAction
  - AcceptChangesDuringFill, ContinueUpdateOnError
  - Fill, Update
  - Eventos: FillError, RowUpdated, RowUpdating
- **CommandBuilder:** Genera automáticamente las propiedades de comando de un DataAdapter u obtiene de un procedimiento almacenado información acerca de parámetros
  - GetInsertCommand, GetUpdateCommand, GetDeleteCommand
- **TableAdapters:** Comunican un DataSet con una base de datos. Se crean con el Diseñador de DataSet dentro de los conjuntos de datos con establecimiento inflexible de tipos con una colección de DataAdapter.

© JMA 2016. All rights reserved

## Accediendo a datos: Desconectado

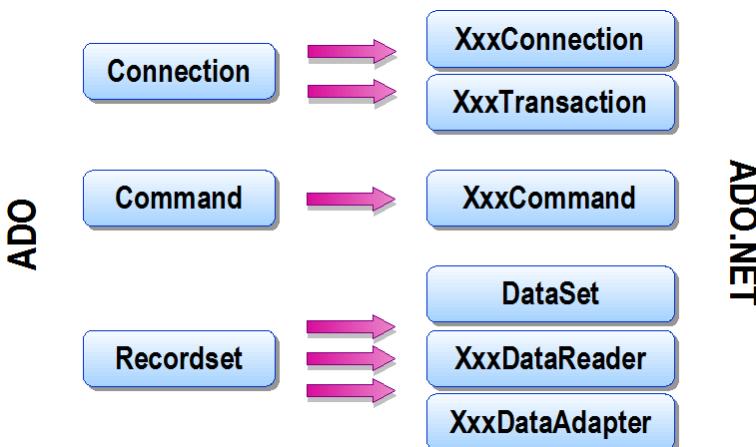
En un escenario desconectado, los recursos no se mantienen en el servidor mientras los datos se procesan:

1. Abrir Conexión
2. Llenar DataSet mediante DataAdapter
3. Cerrar Conexión
4. Procesar DataSet
5. Abrir Conexión
6. Actualizar fuente de datos mediante DataAdapter
7. Cerrar Conexión



© JMA 2016. All rights reserved.

## ADO.NET vs. ADO



© JMA 2016. All rights reserved.

## A partir del ADO.NET 2.0

- API independiente del proveedor ADO.NET
  - Modelada bajo el patrón “Abstract Factory”
- Operaciones Asincrónicas
  - Permite ejecutar comandos contra la base de datos de manera asincrónica no bloqueante
  - Incorpora los métodos BeginExecute.
- Multiple Active Result Sets (MARS)
  - Permite tener múltiples DataReaders abiertos sobre la misma conexión

© JMA 2016. All rights reserved

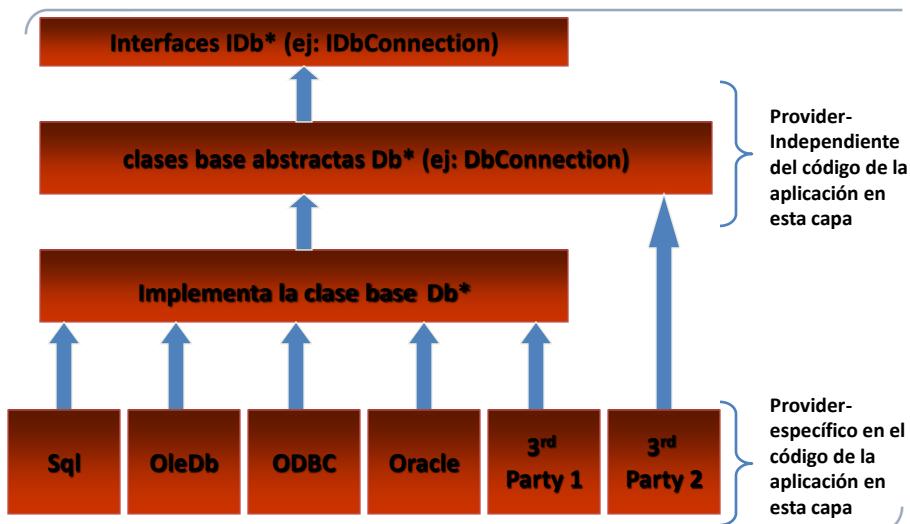
## API Independiente

- Namespace System.Data.Common

DbCommand	DbCommandBuilder	DbConnection
DataAdapter	DbDataAdapter	DbDataReader
DbParameter	DbParameterCollection	DbTransaction
DbProviderFactory	DbProviderFactories	DbException

© JMA 2016. All rights reserved

## API Independiente



© JMA 2016. All rights reserved.

## Abstract Factory

```

string providerName = ConfigurationManager.ConnectionStrings[
    Properties.Settings.Default.ConexionBD].ProviderName;
string connectionString = ConfigurationManager.ConnectionStrings[
    Properties.Settings.Default.ConexionBD].ConnectionString;
DbProviderFactory factory =
    DbProviderFactories.GetFactory(providerName);
DbConnection connection = factory.CreateConnection();
connection.ConnectionString = connectionString;
DbCommand cmd = connection.CreateCommand();
cmd.CommandType = ...;
cmd.CommandText = ...;
cmd.Parameters.Add(...);
cmd.Connection.Open();
DbDataReader dr = cmd.ExecuteReader()

```

© JMA 2016. All rights reserved.

## Adaptadores dinámicos

```
DbDataAdapter adapter = factory.CreateDataAdapter();
adapter.SelectCommand = SelectCommand;

// Create the DbCommandBuilder.
DbCommandBuilder builder =
    factory.CreateCommandBuilder();
builder.DataAdapter = adapter;

// Get the insert, update and delete commands.
adapter.InsertCommand = builder.GetInsertCommand();
adapter.UpdateCommand = builder.GetUpdateCommand();
adapter.DeleteCommand = builder.GetDeleteCommand();
```

© JMA 2016. All rights reserved.

## ADO.NET 2.0 - DataSet

- Mejoras de performance
  - Mantienen indices internos de los registros de sus DataTables
- Serialización binaria del contenido
  - El DataSet 1.x es siempre serializado a XML
    - Bueno para integrar datos, malo en performance
  - El DataSet 2.0 soporta serialización binaria
    - Rápido y compacto
    - DataSet.RemotingFormat = SerializationFormat.Binary

© JMA 2016. All rights reserved.

## ADO.NET 2.0 - DataTable

- Operaciones comunes del DataSet también disponibles en el DataTable:
  - ReadXml, ReadXmlSchema, WriteXml, WriteXmlSchema, Clear, Clone, Copy, Merge, GetChanges
- DataTable es auto-serializable:
  - Buen mecanismo para transmitir datos en una aplicación distribuida

© JMA 2016. All rights reserved

## Tipo de dato XML en el DataSet

- DataTable acepta columnas de tipo XML
  - System.Data.SqlTypes.SqlXml
- Expuestas como una instancia de XPathDocument
- Pueden accederse vía XmlReader
- Facilidades para trabajar con documentos XML como un conjunto de valores

© JMA 2016. All rights reserved

## ADO.NET 2.0 - Actualizaciones Batch

- ADO.NET 2.0 permite ejecutar múltiples instrucciones SQL sobre una base de datos de forma batch, usando el sp\_executesql
- Reduce tráfico de red
- DataAdapter.UpdateBatchSize = batch\_size
- Trabaja con transacciones
- Trabaja con los proveedores para SQL Server y Oracle

© JMA 2016. All rights reserved

## INTRODUCCIÓN AL ENTITY FRAMEWORK

© JMA 2016. All rights reserved

# Introducción

- ADO.NET Entity Framework permite a los desarrolladores crear aplicaciones de acceso a datos programando con un esquema conceptual en lugar de programar directamente con un esquema de almacenamiento relacional.
- ¿Qué conseguimos?
  - Reducimos cantidad de código
  - Simplificamos el mantenimiento de este tipo de aplicaciones

© JMA 2016. All rights reserved

## La Plataforma de Acceso a Datos

```
// Create a connection with the AdventureWorks connection string.
using (SqlConnection conn = new SqlConnection(
    ConfigurationManager.ConnectionStrings["AdventureWorks"].ConnectionString))
{
    conn.Open();

    // Create a command to join Customer and CustomerContactInfo tables.
    SqlCommand command = conn.CreateCommand();
    command.CommandText = @"
        SELECT cust.FirstName, cust.LastName, contact.EmailAddress
        FROM [dbo].[Customer] AS cust
        JOIN [dbo].[CustomerContactInfo] AS contact
        ON cust.CustomerID = contact.CustomerID
        WHERE cust.MiddleName IS NULL";

    // Execute the command and obtain a data reader.
    using (SqlDataReader reader = command.ExecuteReader(
        CommandBehavior.SequentialAccess))
    {
        while (reader.Read())
        {
            // Write a response line using values from the reader.
            Response.Write(String.Format("<p>{0}<\t{1}<\t{2}</p>",
                reader["FirstName"],
                reader["LastName"],
                reader["EmailAddress"]));
        }
    }
}
```

© JMA 2016. All rights reserved

## La Plataforma de Acceso a Datos

```
using (AdventureWorksModel model = new AdventureWorksModel())
{
    var query = from c in model.Customer
                where c.MiddleName == null
                select new {
                    FirstName = c.FirstName,
                    LastName = c.LastName,
                    EmailAddress = c.EmailAddress };

    foreach (var c in query)
    {
        Response.Write(String.Format("<p>{0}\t{1}\t{2}</p>",
            c.FirstName,
            c.LastName,
            c.EmailAddress));
    }
}
```

© JMA 2016. All rights reserved.

## Beneficios de la Plataforma

### Language Integrated Query

- Consultas embebidas en lenguaje
- Sintaxis nativa en C# y VB
- Soporte a múltiples fuentes de datos

### Entity Data Model (EDM)

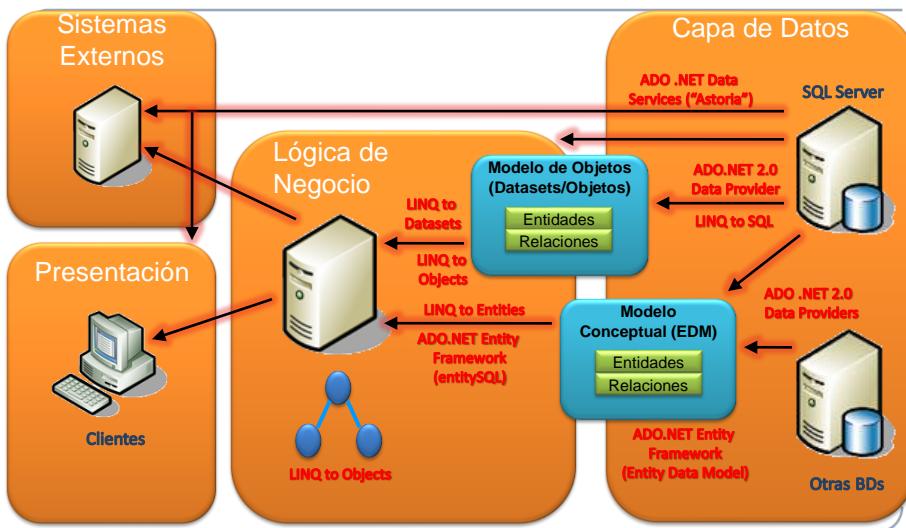
- Eleva el nivel de abstracción
- “Artefactos” reutilizables
- Crea un ecosistema de servicios

### Acceso a la Información

- Compromiso a una amplia plataforma
- Compromiso continuado a ODBC
- Soporte para terceros en EF (JDBC, ...)

© JMA 2016. All rights reserved.

## Como encajan las piezas



© JMA 2016. All rights reserved

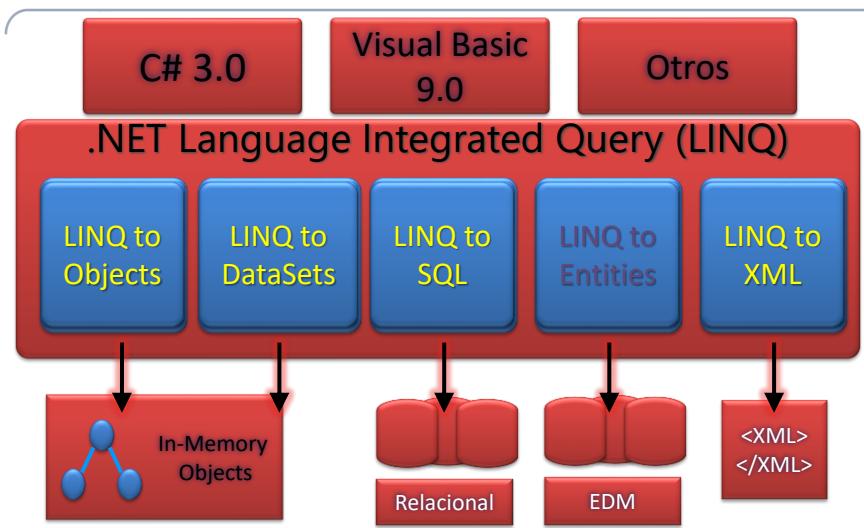
## Language INtegrated Query

- Lenguaje de consultas único
- Datos == Objetos
- Funciona contra objetos, relacional y XML

```
using (AdventureWorksModel model = new AdventureWorksModel())
{
    var query = from c in model.Customer
                where c.MiddleName == null
                select new {
                    FirstName = c.FirstName,
                    LastName = c.LastName,
                    EmailAddress = c.EmailAddress };
```

© JMA 2016. All rights reserved

## Language INtegrated Query



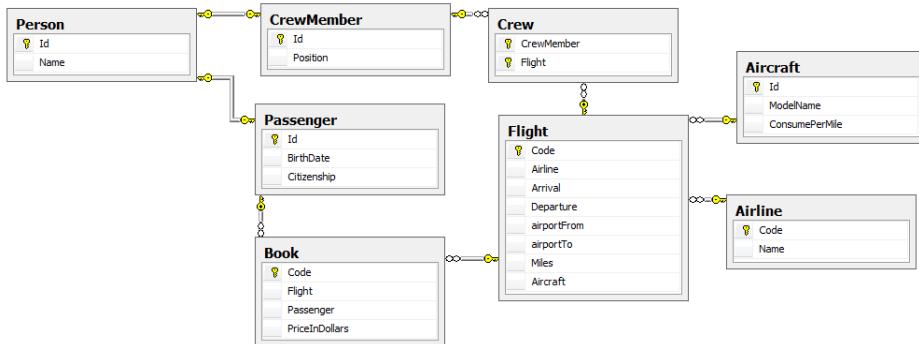
© JMA 2016. All rights reserved

## ADO.NET Entity Framework

- La siguiente capa del stack de ADO.NET
- Describe tus datos usando un modelo conceptual y ADO.NET hará el resto
  - Herramientas de diseño para el Entity Data Model
  - Mapeo declarativo con la Base de Datos
  - Generación de clases .NET para las entidades de negocio
  - Consulta utilizando LINQ to Entities y Entity SQL
  - Se encarga de las actualizaciones automáticamente con T-SQL o procedimientos

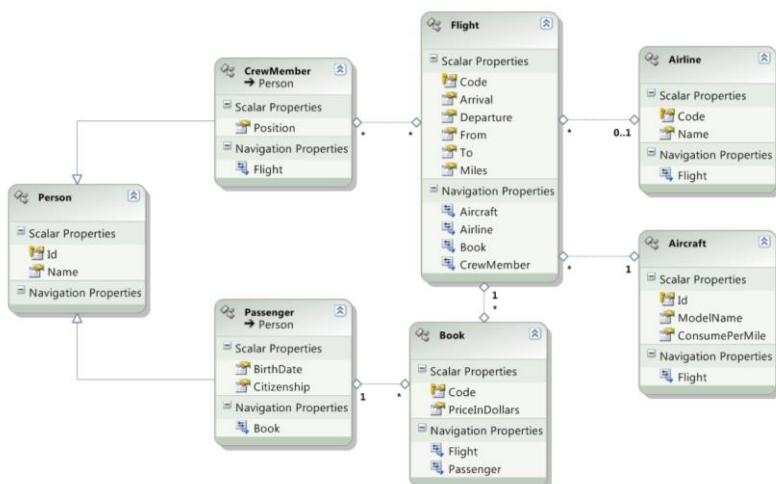
© JMA 2016. All rights reserved

# Modelo entidad/relación



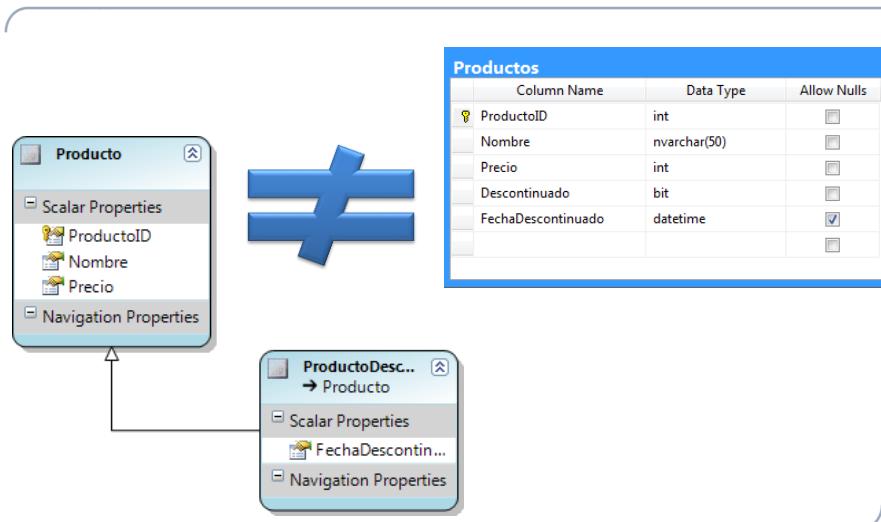
© JMA 2016. All rights reserved

# Modelo de dominio



© JMA 2016. All rights reserved

## Impedance Mismatch



© JMA 2016. All rights reserved.

## Discrepancia del Paradigma

- Bauer & King (Bauer, C. & King, G. 2007) presentan una lista de los problemas de discrepancia en los paradigmas objeto/relacional
  - Problemas de granularidad
    - Los objetos pueden definir clases con diferentes niveles de granularidad.
      - Cuanto más fino las clases de grano (Direcciones), éstas pueden ser embebida en las clases de grano grueso (Usuario)
    - En cambio, el sistema de tipo de la base de datos SQL son limitados y la granularidad se puede aplicar sólo en dos niveles
      - en la tabla (tabla de usuario) y el nivel de la columna (columna de dirección)
  - Problemas de subtipos
    - La herencia y el polimorfismo son las características básicas y principales del lenguaje de programación O-O.
    - Los motores de base de datos SQL en general, no son compatibles con subtipos y herencia de tablas.

© JMA 2016. All rights reserved.

# Discrepancia del Paradigma

- Problemas de identidad
  - Los objetos definen dos nociones diferentes de identidad:
    - Identidad de objeto o referencia (equivalente a la posición de memoria, comprobar con un `==` ).
    - La igualdad como determinado por la aplicación de los métodos `Equals()` .
  - La identidad de una fila de base de datos se expresa como la clave primaria.
  - Ni `Equals()` ni `==` es equivalente a la clave principal.
- Problemas de Asociaciones
  - En objetos se representa a las asociaciones mediante utilizan referencias a objetos
    - Las asociaciones entre objetos son punteros unidireccionales.
    - Relación de pertenencia → Elementos contenidos
    - Modelo de composición (colecciones) → Modelo jerárquico
  - Las asociaciones en BBDD están representados mediante la migración de claves.
    - Todas las asociaciones en una base de datos relacional son bidireccional

© JMA 2016. All rights reserved

## ¿Por qué un ORM?

- Porque trabajamos en lenguajes orientados a objetos
- Porque los datos se almacenan siguiendo un modelo relacional.
- Los mismos datos que “viven” en la base de datos relacional, se representan de una forma totalmente diferente en la aplicación
- Los desarrolladores desperdician mucho tiempo y energía escribiendo código para traducir de objetos a tablas y viceversa
- Para disminuir el “impedance mismatch” entre una aplicación orientada a objetos y una base de datos relacional
- Aumentar el nivel de abstracción en el acceso a los datos

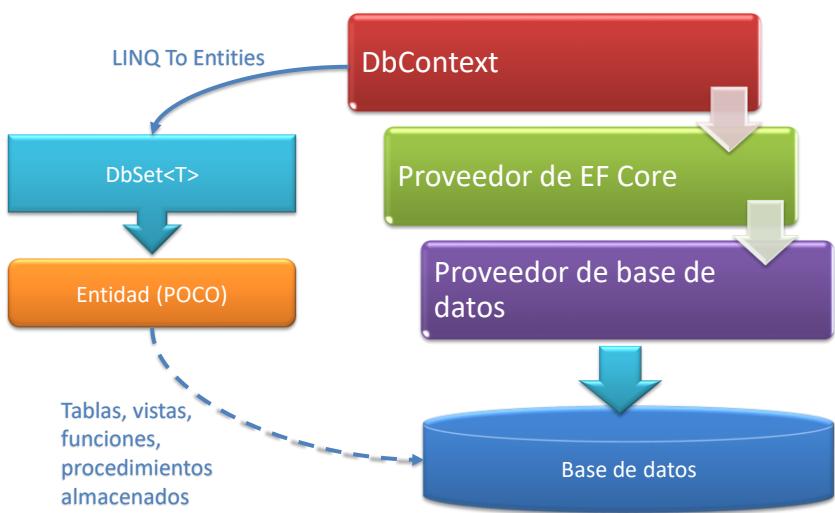
© JMA 2016. All rights reserved

# Entity Framework Core (EF Core)

- Entity Framework Core (EF Core) es un mapeador objetos-relaciones (ORM). Un ORM proporciona una capa entre el modelo de dominio que se implementa en el código y una base de datos. EF Core es una API de acceso a datos que permite interactuar con la base de datos mediante objetos POCO (Plain Old CLR Object) de .NET, que no dependen de un framework, y sintaxis de LINQ fuertemente tipada.
- En EF Core, la base de datos se abstracta detrás de los POCO de .NET. EF Core controla la interacción directa con la base de datos subyacente. Al usar esta API, puede dedicar menos tiempo a traducir solicitudes a y desde la base de datos y escribir SQL, y más tiempo a centrarse en la lógica de negocios importante.
- Con EF Core, puede hacer lo siguiente:
  - Cargar datos como objetos (entidades) de C#.
  - Agregar, modificar y eliminar datos llamando a métodos en las entidades.
  - Asignar varias tablas de base de datos a una sola entidad de C#.
  - Controlar los problemas de simultaneidad que surgen cuando varios usuarios intentan actualizar simultáneamente el mismo registro.
  - Usar la sintaxis LINQ (System.Linq) fuertemente tipada para consultar la base de datos.
  - Acceder a varias bases de datos, como SQL Server, SQLite, PostgreSQL, MySQL, etc.
  - Compilar el modelo de dominio a partir de una base de datos existente.
  - Administrar el esquema de base de datos en función del modelo de dominio.
  - Confirmar los cambios en gráficos de objetos complejos, profundos o anchos de entidades relacionadas con una sola llamada de método.

© JMA 2016. All rights reserved

## Arquitectura de EF Core



© JMA 2016. All rights reserved

# Arquitectura de EF Core

- DbContext representa una sesión con la base de datos y se puede usar para consultar y guardar instancias de las entidades. DbContext es una combinación de los patrones de unidad de trabajo y repositorio. DbContext proporciona métodos para configurar opciones, cadenas de conexión, registros y el modelo usado para asignar el dominio a la base de datos. Las clases que derivan de DbContext:
  - Representan una sesión activa con la base de datos.
  - Guardan y consultan instancias de entidades.
  - Incluyen propiedades de tipo DbSet<T> que representan tablas (vistas, funciones, ...) en la base de datos.
- El proveedor de EF Core traduce los cambios de grafo de objeto a código SQL.
- El proveedor de base de datos es una biblioteca de complementos que se ha diseñado para un motor de base de datos específico, como SQL Server, Azure Cosmos DB o PostgreSQL. Traduce las llamadas de método y las consultas LINQ al dialecto SQL nativo de la base de datos. Amplía EF Core para habilitar una funcionalidad que es única para el motor de base de datos.

© JMA 2016. All rights reserved

## Instalación

- Obtener Entity Framework Core
  - dotnet add package Microsoft.EntityFrameworkCore.SqlServer
  - Install-Package Microsoft.EntityFrameworkCore.SqlServer
- Obtención de las herramientas de la CLI de .NET Core
  - dotnet tool install --global dotnet-ef
  - dotnet add package Microsoft.EntityFrameworkCore.Design
  - Install-Package Microsoft.EntityFrameworkCore.Tools
- Creación de la base de datos
  - dotnet ef migrations add InitialCreate
  - dotnet ef database update

© JMA 2016. All rights reserved

# Proveedores de bases de datos

Sistema de base de datos	Paquete
SQL Server y SQL Azure	<a href="#">Microsoft.EntityFrameworkCore.SqlServer</a>
SQLite	<a href="#">Microsoft.EntityFrameworkCore.Sqlite</a>
Azure Cosmos DB	<a href="#">Microsoft.EntityFrameworkCore.Cosmos</a>
PostgreSQL	<a href="#">Npgsql.EntityFrameworkCore.PostgreSQL*</a>
MySQL	<a href="#">Pomelo.EntityFrameworkCore.MySql*</a>
Base de datos en memoria de EF Core	<a href="#">Microsoft.EntityFrameworkCore.InMemory</a>

<https://learn.microsoft.com/es-es/ef/core/providers/?tabs=dotnet-core-cli>

© JMA 2016. All rights reserved.

## DbContext

- La duración de DbContext comienza cuando se crea la instancia y finaliza cuando la instancia se elimina. Una instancia de DbContext está diseñada para usarse para una única unidad de trabajo. Esto significa que la duración de una instancia de DbContext suele ser muy breve.
- Una unidad de trabajo hace un seguimiento de todas las acciones que realiza durante una transacción comercial que pueden afectar a la base de datos. Cuando ha terminado, determina todo lo que se debe hacer para modificar la base de datos como resultado de su trabajo. Una unidad de trabajo típica al utilizar Entity Framework Core (EF Core) implica lo siguiente:
  - Creación de una instancia de DbContext.
  - Seguimiento de las instancias de entidad por el contexto. Seguimiento de las entidades mediante devolución desde una consulta o una adición o asociación al contexto
  - Se realizan cambios en las entidades sometidas a seguimiento según sea necesario para implementar la regla empresarial.
  - Se llama a SaveChanges o SaveChangesAsync, EF Core detecta los cambios realizados, genera el SQL y lo manda a la base de datos.
  - Se elimina la instancia de DbContext (garantiza que se liberen los recursos).

© JMA 2016. All rights reserved.

# DbContextOptions

- El punto inicial de toda la configuración de DbContext es DbContextOptionsBuilder. Hay tres maneras de obtener este generador:
  - Con AddDbContext y métodos relacionados (ASP.NET Core)
  - Sobre escribiendo OnConfiguring
  - Explícitamente con new

```
var contextOptions = new
DbContextOptionsBuilder<ApplicationDbContext>()
.UseSqlServer(@"Server=(localdb)\mssqllocaldb;Database=Test;
").Options;
using(var context = new ApplicationDbContext(contextOptions)) {
```

- Los métodos de Use\* son métodos de extensión implementados por el proveedor de bases de datos. Esto significa que el paquete NuGet del proveedor de bases de datos debe estar instalado para poder usar el método de extensión.

© JMA 2016. All rights reserved

# Modelo

- EF Core usa un modelo de metadatos para describir cómo se asignan los tipos de entidad de la aplicación a la base de datos subyacente. Este modelo se crea con un conjunto de convenciones: heurística que busca patrones comunes. Después, el modelo se puede personalizar mediante atributos de asignación (también conocidos como anotaciones de datos) o llamadas a los métodos ModelBuilder(también conocido como API fluida) en OnModelCreating; ambos reemplazarán la configuración que realizan las convenciones.
- La mayoría de la configuración se puede aplicar a un modelo que tenga como destino cualquier almacén de datos. Los proveedores también pueden habilitar la configuración específica de un almacén de datos determinado, así como omitir la configuración que no es compatible o no es aplicable. Para obtener documentación sobre la configuración específica del proveedor, vea la sección Proveedores de bases de datos.

© JMA 2016. All rights reserved

# Esquemas de base de datos

- EF Core proporciona dos métodos principales para mantener sincronizados el esquema de la base de datos y el modelo de EF Core.
- Para elegir entre los dos, decida si es el modelo de EF Core o el esquema de la base de datos es un origen de confianza.
  - Si quiere que el modelo de EF Core sea el origen verdadero, use Migraciones. Al realizar cambios en el modelo de EF Core, este método aplica de forma incremental los cambios de esquema correspondientes a la base de datos para que siga siendo compatible con el modelo de EF Core.
  - Si quiere que el esquema de la base de datos sea el origen verdadero, use Ingeniería inversa. Este método permite aplicar la técnica de scaffolding a un elemento DbContext y a las clases de tipo de entidad mediante la aplicación de ingeniería inversa al esquema de la base de datos de un modelo de EF Core.

© JMA 2016. All rights reserved

## Code First

- Instalación de Entity Framework Core
  - Install-Package Microsoft.EntityFrameworkCore.SqlServer
  - Install-Package Microsoft.EntityFrameworkCore.Tools
- Creación de las entidades (modelos) en la carpeta Models (nombres en singular)
- Creación de una clase de contexto de base de datos (hereda de DbContext o uno de sus herederos) en la carpeta Data, agregando un DbSet por entidad (nombres en plural)
 

```
public class ApplicationDbContext : IdentityDbContext {
    public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
        : base(options) {
    }
    public DbSet<Producto> Productos { get; set; }
    protected override void OnModelCreating(ModelBuilder modelBuilder) {
        modelBuilder.Entity<Producto>(entity => { ... });
    }
}
```
- Generar y revisar el archivo de migración:  
Add-Migration InitialCreate
- Actualizar (o crear) la base de datos  
Update-Database

© JMA 2016. All rights reserved

# Database First

- Instalación de Entity Framework Core
  - Install-Package Microsoft.EntityFrameworkCore.SqlServer
  - Install-Package Microsoft.EntityFrameworkCore.Tools
- La ingeniería inversa: Database First
  - get-help scaffold-dbcontext –detailed
- Generar clases:
  - Scaffold-DbContext "Data Source=.;Initial Catalog=AdventureWorksLT2017;Integrated Security=True;Connect Timeout=30;Encrypt=False;TrustServerCertificate=False;ApplicationIntent=ReadWrite;MultiSubnetFailover=False" Microsoft.EntityFrameworkCore.SqlServer -DataAnnotations -ContextDir Infrastructure.Data -UnitOfWork -Context TiendaDbContext -OutputDir Domain.Entities -Schemas SalesLT

© JMA 2016. All rights reserved

# Database First

- La cadena de conexión en appsettings.json:
 

```
{
  "ConnectionStrings": {
    "TiendaConnection": "Data Source=.;Initial Catalog=AdventureWorksLT2017;Integrated Security=True;Connect Timeout=30;Encrypt=False;TrustServerCertificate=False;ApplicationIntent=ReadWrite;MultiSubnetFailover=False"
  }
}
```
- La inyección de dependencias en Startup.ConfigureServices:
 

```
services.AddDbContext<TiendaDbContext>(options => options.UseSqlServer(
    Configuration.GetConnectionString("TiendaConnection")));
builder.Services.AddDbContext<TiendaDbContext>(options =>
    options.UseSqlServer(
        builder.Configuration.GetConnectionString("TiendaConnection")));
```
- Eliminar el método OnConfiguring con la cadena de conexión de la clase TiendaDbContext.

© JMA 2016. All rights reserved

# Database First

- Instalación de las herramientas
  - dotnet tool install --global dotnet-ef
- Comprobar la instalación:
  - dotnet ef
- Instalación de Entity Framework Core
  - Install-Package Microsoft.EntityFrameworkCore.SqlServer
  - Install-Package Microsoft.EntityFrameworkCore.Tools
- Generar clases:
  - dotnet ef dbcontext scaffold "Data  
Source=(localdb)\MSSQLLocalDB;Initial Catalog=Curso"  
Microsoft.EntityFrameworkCore.SqlServer --context-dir  
Data --output-dir Models --data-annotations

© JMA 2016. All rights reserved

## Plantillas personalizadas de ingeniería inversa

- Aunque en la utilización de técnicas de ingeniería inversa, Entity Framework Core se esfuerza por aplicar scaffolding a un buen código de uso general que se puede usar en una variedad de tipos de aplicaciones y usa convenciones de codificación comunes para una apariencia coherente y familiar. A veces, sin embargo, es deseable un código más especializado y estilos de codificación alternativos, se puede personalizar el código con scaffolding mediante plantillas de texto T4.
  - dotnet new install Microsoft.EntityFrameworkCore.Templates
  - dotnet new ef-templates
- El último comando agrega los siguientes archivos al proyecto.
 

```
CodeTemplates/
  EFCore/
    DbContext.t4
    EntityType.t4
```
- La plantilla DbContext.t4 se usa para aplicar scaffolding a una clase DbContext para la base de datos y la plantilla EntityType.t4 se usa para aplicar scaffolding a las clases de tipo de entidad para cada tabla y vista de la base de datos.

© JMA 2016. All rights reserved

# Entidades

- Una entidad es cualquier objeto del dominio que mantiene un estado y comportamiento más allá de la ejecución de la aplicación y que necesita ser distinguido de otro que tenga las mismas propiedades y comportamientos.
- Las entidades encapsulan los conceptos del negocio. Una entidad puede ser un objeto con métodos o puede ser un conjunto de funciones y estructuras de datos. No importa siempre que las entidades puedan ser utilizadas por muchas aplicaciones diferentes en la empresa.
- Estas entidades son los objetos de dominio de la aplicación. Encapsulan las reglas más generales y de alto nivel. Son los menos propensos a cambiar cuando algo externo cambia. Por ejemplo, no esperaría que estos objetos se vieran afectados por un cambio en la navegación de la página o la seguridad. Ningún cambio operativo en una aplicación en particular debería afectar la capa de la entidad.
- Las entidades exponen a través de propiedades los datos persistentes. Una entidad es mas que una estructura de datos (estado), puede contener métodos y eventos que implementen las reglas de negocio que afectan a la entidad individual.

© JMA 2016. All rights reserved

# Entidades

- En EF Core, las entidades son objetos POCO (Plain Old CLR Object) de .NET, que no dependen del framework, no necesitan heredar o implementar interfaces específicos.
- EF Core utiliza las convenciones de nombres para mapear las clases a tablas y las propiedades a columnas.
- Se pueden aplicar determinados atributos (conocidos como anotaciones de datos) a las clases y propiedades que reemplazarán a las convenciones.
  - [Table("blogs", Schema = "blogging")]: Nombre y esquema de la tabla (clase)
  - [NotMapped]: Evita que una propiedad pública con un captador y un establecedor se incluyan en el modelo.
  - [Column(...)]: Permite establecer el nombre, tipo y orden de la columna.
  - [Required],[MaxLength(500)], [Precision(14, 2)], [Unicode(false)]: Establecen restricciones de not null, longitud máxima, precisión y escala y Unicode para representar datos de texto.
  - [Key] (propiedad), [PrimaryKey(...)] (clase): Configurar la clave principal.
  - [DatabaseGenerated(DatabaseGeneratedOption.Identity)]: Configuración explícita de la generación de valores (no solo claves).

© JMA 2016. All rights reserved

# Relaciones en modelos de objetos

- Las relaciones en objetos se establecen mediante asociaciones, referencias o colecciones (HashSet), y no mediante migraciones de claves.
- La asignación de relaciones de EF Core consiste en asignar la representación de clave principal o clave externa usada en una base de datos relacional a las referencias entre los objetos usados en un modelo de objetos.
- En el sentido más básico, esto implica lo siguiente:
  - Agregar una propiedad de clave principal a cada tipo de entidad.
  - Agregar una propiedad de clave externa a un tipo de entidad.
  - Asociar las referencias entre los tipos de entidad con las claves principales y externas para formar una única configuración de relación.
- Las relaciones de EF Core se definen mediante claves externas. Las navegaciones se superponen sobre claves externas a fin de proporcionar una vista natural orientada a objetos para leer y manipular relaciones. Mediante las navegaciones, las aplicaciones pueden trabajar con grafos de entidades sin preocuparse de lo que sucede con los valores de claves externas.
  - [ForeignKey] se usa para conectar una propiedad de clave externa con sus navegaciones
  - [InverseProperty] se usa para conectar una navegación con su inversa (solo es necesario cuando hay más de una relación entre los mismos tipos).
  - [DeleteBehavior] se usa para las relaciones opcionales y el comportamiento Cascade para las relaciones obligatorias.

© JMA 2016. All rights reserved

# Modelos de Datos Avanzados

- Tipos complejos
  - Por ejemplo un campo “Dirección”
- Herencias por Jerarquía
  - Una tabla física para todos los tipos con un campo discriminador
- Herencias por SubTipo
  - Una tabla física por cada tipo en la jerarquía con relaciones 1:1 en el modelo relacional
- Herencias por Tipo
  - Múltiples tablas para un mismo tipo
  - Dos tablas para una misma entidad
- Procedimientos Almacenados
- Secuencias

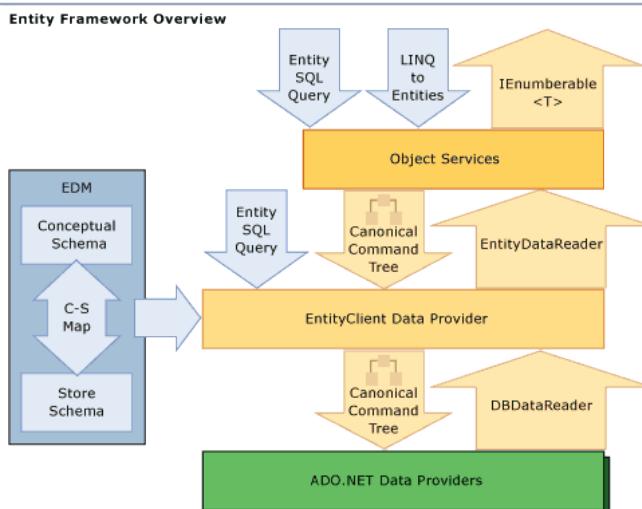
© JMA 2016. All rights reserved

# LINQ To Entities

- Entity Framework Core usa Language Integrated Query (LINQ) para consultar datos de la base de datos. LINQ permite usar C# (o el lenguaje .NET que prefiera) para escribir consultas fuertemente tipadas.
- Uso el contexto derivado y las clases de entidad para hacer referencia a los objetos de base de datos. Incluir un DbSet de un tipo en el contexto significa que se incluye en el modelo de EF Core. DbSet se puede usar para las consultas LINQ.
- EF Core pasa una representación de la consulta LINQ al proveedor de la base de datos.
- A su vez, los proveedores de la base de datos la traducen al lenguaje de la consulta específico para la base de datos (por ejemplo, SQL para una base de datos relacional).
- Las consultas siempre se ejecutan en la base de datos incluso si las entidades devueltas en el resultado ya existen en el contexto. Los resultados de una consulta LINQ en un DbSet contendrán los resultados devueltos de la base de datos y pueden no reflejar los cambios realizados en el contexto que no se han conservado en la base de datos.

© JMA 2016. All rights reserved

## Arquitectura de LINQ To Entities



© JMA 2016. All rights reserved

## LINQ To Entities

- Como norma general, Entity Framework Core intenta evaluar una consulta en el servidor lo máximo posible. EF Core convierte partes de la consulta en parámetros, que se pueden evaluar en el lado cliente.
- El resto de la consulta (junto con los parámetros generados) se proporciona al proveedor de base de datos para determinar la consulta de base de datos equivalente que se va a evaluar en el servidor.
- EF Core admite la evaluación de cliente parcial en la proyección de nivel superior (fundamentalmente, la última llamada a `Select()`). Si la proyección de nivel superior de la consulta no se puede traducir en el servidor, EF Core capturará los datos necesarios del servidor y evaluará las partes restantes de la consulta en el cliente. Si EF Core detecta una expresión, en cualquier lugar que no sea la proyección de nivel superior, que no se puede traducir en el servidor, inicia una excepción en tiempo de ejecución.

© JMA 2016. All rights reserved

## LINQ To Entities

- Diferencias con LINQ To Objects y To XML
- Las consultas tienen el tipo **ObjectQuery<T>**
  - Implementa `IQueryable<T>` y no `IEnumerable<T>`
  - Necesita Árboles de expresión para construir el SQL final.
- Cuando se enumeran los resultados:
  - `query1.FirstOrDefault()`, `query1.ToList()` ...
  - Se ejecuta la consulta SQL y devuelve un `IEnumerable<T>`
- No están disponibles todos los operadores de Linq To Objects o To XML

© JMA 2016. All rights reserved

# Operadores disponibles

Expresión de consulta de Linq	Where(), OfType(), Select(), SelectMany(), Zip(), OrderBy(), ThenBy(), OrderByDescending(), ThenByDescending(), Reverse(), GroupBy(), Join(), GroupJoin(), DefaultIfEmpty()
Partición	Take(), TakeWhile(), Skip(), SkipWhile(), SkipLast(), Chunk()
Conjunto	Distinct(), Union(), Intersect(), Except(), DistinctBy(), UnionBy(), IntersectBy(), ExceptBy()
Conversión	ToArray(), ToList(), ToDictionary(), ToLookup(), AsEnumerable(), AsQueryable(), Cast<T>(), OfType<T>()
Combinación	Append(), Concat(), Prepend()
Cuantificación	Any(), All(), Contains()
Elementos	First(), Last(), ElementAt(), Single(), {método}OrDefault()
Agregados	Aggregate(), Count(), LongCount(), Max(), Min(), Sum(), Average()

© JMA 2016. All rights reserved

## Cómo hacer una consulta

- Establecer el contexto
  - Using (FlightContext ctx = new FlightContext()) { ... }
- Dentro del contexto construir la consulta
  - Obtener los IQueryables<T> del Contexto
  - IQueryables<Flight> query =
 

```
from f in ctx.Flights
where p.To=="MAD"
select f;
```
- Ejecutar la consulta
  - Con un operador de conversión
    - query.ToList(); query.FirstOrDefault() ...

© JMA 2016. All rights reserved

## Datos relacionados

- Entity Framework Core permite usar las propiedades de navegación del modelo para cargar las entidades relacionadas. Los tres patrones de O/RM comunes son:
  - Carga diligente (JOIN): los datos relacionados se cargan desde la base de datos como parte de la consulta inicial. Se pueden filtrar, incluir automáticamente.
 

```
dbContext.Blogs.Include(blog => blog.Posts).ThenInclude(post => post.Author)
modelBuilder.Entity<Order>().Navigation(e => e.Details).AutoInclude();
```
  - Carga explícita (múltiples): los datos relacionados se cargan de manera explícita desde la base de datos más adelante.
 

```
dbContext.Entry(blog).Collection(b => b.Posts).Load();
dbContext.Entry(blog).Reference(b => b.Owner).Load();
var goodPosts = Entry(blog).Collection(b => b.Posts)
    .Query().Where(p => p.Rating > 3).ToList();
```
  - Carga diferida (múltiples): los datos relacionados se cargan de manera transparente desde la base de datos cuando se accede a la propiedad de navegación, requiere UseLazyLoadingProxies o el servicio ILazyLoader.

© JMA 2016. All rights reserved

## Consultas divididas

- Al trabajar con bases de datos relacionales, EF carga entidades relacionadas mediante la introducción de JOIN en una sola consulta. Aunque los JOIN son bastante estándar al usar SQL, pueden crear problemas de rendimiento significativos, que a veces se denomina explosión cartesiana, puede provocar que grandes cantidades de datos se transfieran involuntariamente al cliente, si se usan incorrectamente. pueden crear otro tipo de problema con los JOIN es la duplicación de datos, varias instancias de la misma fila.
- Para solucionar los problemas de rendimiento descritos anteriormente, EF le permite especificar que una consulta LINQ determinada debe dividirse en varias consultas SQL. En lugar de instrucciones JOIN, las consultas divididas generan una consulta SQL adicional por cada navegación de colección incluida:
 

```
context.Blogs.Include(blog => blog.Posts).AsSplitQuery().ToList();
```
- Si bien las consultas divididas evitan las incidencias de rendimiento asociadas a las instrucciones JOIN y la explosión cartesiana, también existen algunas desventajas.

© JMA 2016. All rights reserved

## Consultas SQL

- Entity Framework Core le permite descender hasta las consultas SQL cuando trabaja con una base de datos relacional. Las consultas SQL son útiles si la consulta que le interesa no se puede expresar mediante LINQ, o bien si una consulta LINQ hace que EF genere código SQL ineficaz. Las consultas SQL pueden devolver tipos de entidad normales o tipos de entidad sin clave que forman parte del modelo.

```
var blogs = dbContext.Blogs
    .FromSql($"SELECT * FROM dbo.Blogs WHERE {propertyName} =
{propertyValue}").ToList();
var blogs = dbContext.Blogs
    .FromSql($"EXECUTE dbo.GetMostPopularBlogsForUser {user}").ToList();
var count = dbContext.Database
    .SqlQuery<int>($"SELECT count(*) as Value FROM [Blogs]").Single();
var updates= dbContext.Database.ExecuteSql($"UPDATE Blogs SET Url = NULL");
```

- Los métodos `FromSql` y `FromSqlInterpolated` están protegidos contra la inyección de código SQL y siempre integran los datos de parámetros como un parámetro SQL independiente. Aun así, el método `FromSqlRaw` puede ser vulnerable a ataques por inyección si se usa incorrectamente.

© JMA 2016. All rights reserved

## Seguimiento

- El comportamiento de seguimiento controla si Entity Framework Core mantiene información sobre una instancia de entidad en su herramienta de seguimiento de cambios. Si se realiza el seguimiento de una entidad, cualquier cambio detectado en ella se conserva en la base de datos durante `SaveChanges`. EF Core también corrige las propiedades de navegación entre las entidades de un resultado de consulta de seguimiento y las entidades que se encuentran en la herramienta de seguimiento de cambios.
- De manera predeterminada, las consultas que devuelven tipos de entidad son consultas de seguimiento.
- Las consultas de no seguimiento son útiles cuando los resultados se usan en un escenario de solo lectura y su ejecución suele ser más rápida porque no es necesario configurar la información de seguimiento de cambios.

```
var blogs = context.Blogs.AsNoTracking().ToList();
```

© JMA 2016. All rights reserved

## SaveChanges

- `DbContext.SaveChanges()` es una de las dos técnicas para guardar los cambios en la base de datos con EF. Con este método, se realizan uno o varios cambios supervisados (agregar, actualizar, eliminar) y, a continuación, aplicar esos cambios llamando al método `SaveChanges`.
- Se usa el método `DbSet< TEntity >.Add` para agregar nuevas instancias de las clases de entidad (INSERT):
 

```
dbContext.Blogs.Add(new Blog { Url = "http://example.com" });
```
- EF detecta automáticamente los cambios hechos en una entidad existente de la que hace seguimiento el contexto (UPDATE):
 

```
var blog = context.Blogs.Single(b => b.BlogId == 1);
blog.Url = "http://example.com/blog";
```
- Se usa el método `DbSet< TEntity >.Remove` para eliminar instancias de las clases de entidad (DELETE):
 

```
dbContext.Blogs.Remove(blog);
```
- Los cambios son en el contexto hasta que se envían a la base de datos con `SaveChanges`, que puede combinar múltiples operaciones.
 

```
dbContext.SaveChanges();
```

© JMA 2016. All rights reserved.

## SaveChanges

- Una entidad no recuperada en la instancia del `DbContext` no cuenta con seguimiento de cambios. Hay que agregarla al contexto para que comience a realizar el seguimiento de la entidad y las entradas especificadas accesibles desde la entidad. Con `Attach` se agrega con estado `Unchanged` y con `Update` se agrega con estado `Modified`.
 

```
var blog = new Blog { BlogId = 1, Url = "http://example.com" };
dbContext.Blogs.Attach(blog); // Unchanged
dbContext.Blogs.Update(blog); // Modified
dbContext.Entry(blog).State = EntityState.Deleted;
```
- En el caso de los tipos de entidad con claves generadas, si una entidad tiene su valor de clave principal establecido, se realizará un seguimiento en el estado `Unchanged` o `Modified`. Si no se establece el valor de la clave principal, se realizará el seguimiento en el estado `Added`.

© JMA 2016. All rights reserved.

## Grafo de entidades

- SaveChanges propaga los cambios en las entidades relacionadas.
- Si se crean varias entidades relacionadas, agregar una de ellas al contexto hará que las otras también se agreguen.
- Si hace referencia a una entidad nueva desde la propiedad de navegación de una entidad a la que el contexto ya hace seguimiento, se detectará la entidad y se insertará en la base de datos.
- Si cambia la propiedad de navegación de una entidad, los cambios correspondientes se harán en la columna de clave externa de la base de datos.
- Para quitar una relación, se establece null en la referencia de navegación o se quita la entidad relacionada de una navegación de colección. Quitar una relación puede tener efectos secundarios en la entidad dependiente, según el comportamiento de eliminación en cascada que esté configurado en la relación. Las eliminaciones en cascada son necesarias cuando una entidad dependiente o secundaria ya no se puede asociar a su entidad primaria actual. En las relaciones opcionales se puede establecer la propiedad de clave externa a null.

© JMA 2016. All rights reserved

## Simultaneidad

- EF Core implementa la simultaneidad optimista, que supone que los conflictos de simultaneidad son relativamente poco frecuentes. A diferencia de los enfoques pesimistas, que bloquean los datos por adelantado y después los modifican, la simultaneidad optimista no efectúa bloqueos, sino que se ocupa de que no se guarde la modificación de datos si los datos han cambiado desde que se consultaron. Este error de simultaneidad se notifica a la aplicación, que lo trata en consecuencia, posiblemente mediante el reintento de toda la operación en los nuevos datos.
- En EF Core, la simultaneidad optimista se implementa configurando una propiedad como un token de simultaneidad. El token de simultaneidad se carga y se realiza un seguimiento del mismo cuando se consulta una entidad, al igual que cualquier otra propiedad. A continuación, cuando se realiza una operación de actualización o de eliminación durante la ejecución de SaveChanges(), el valor del token de simultaneidad en la base de datos se compara con el valor original leído por EF Core.

© JMA 2016. All rights reserved

## ExecuteUpdate y ExecuteDelete

- ExecuteUpdate y ExecuteDelete son una manera de guardar datos en la base de datos sin usar el método SaveChanges() y el seguimiento de cambios tradicional de EF.
- Aunque el seguimiento de cambios y SaveChanges() son una manera eficaz de guardar los cambios, requiere que se consulte y realice un seguimiento de todas las entidades que se vayan a modificar o eliminar, que puede penalizar el rendimiento.
- Para admitir escenarios de "actualización masiva", se puede usar:
 

```
dbContext.Blogs.Where(b => b.Rating < 3).ExecuteDelete();
dbContext.Blogs.Where(b => b.Rating < 3)
    .ExecuteUpdate(setters => setters
        .SetProperty(b => b.Visible, false)
        .SetProperty(b => b.Rating, b => b.Rating + 1));
```
- ExecuteUpdate no admite actualmente la referencia a las navegaciones dentro de la expresión lambda SetProperty.
- ExecuteUpdate y ExecuteDelete surten efecto inmediatamente, en el momento en que se invocan, sin esperar al SaveChanges.

© JMA 2016. All rights reserved

## Transacciones

- Las transacciones permiten procesar varias operaciones de base de datos de manera atómica. Si se confirma la transacción, todas las operaciones se aplicaron correctamente a la base de datos. Si se revierte la transacción, ninguna de las operaciones se aplicó a la base de datos.
- De manera predeterminada, si el proveedor de base de datos admite las transacciones, todos los cambios de una llamada sencilla a SaveChanges se aplican a una transacción. Si cualquiera de los cambios presenta un error, la transacción se revertirá y no se aplicará ninguno de los cambios a la base de datos. Esto significa que se garantiza que SaveChanges se complete correctamente o deje sin modificaciones la base de datos si se produce un error.
- Solo debe controlar manualmente las transacciones si los requisitos de la aplicación lo consideran necesario. Se puede usar la API DbContext.Database para iniciar, confirmar y revertir las transacciones:
 

```
using var transaction = dbContext.Database.BeginTransaction();
transaction.Commit();
```
- Los puntos de retorno son puntos dentro de una transacción de base de datos a los que se puede revertir más tarde en caso de que ocurra un error o por cualquier otro motivo.
 

```
transaction.CreateSavepoint("BeforeMoreChanges");
transaction.RollbackToSavepoint("BeforeMoreChanges");
```

© JMA 2016. All rights reserved

## Registros

- Entity Framework Core (EF Core) contiene varios mecanismos para generar registros, responder a eventos y obtener diagnósticos. Cada uno de ellos se adapta a diferentes situaciones, y es importante seleccionar el mecanismo adecuado para cada tarea, incluso cuando puedan funcionar varios.
- Se puede acceder a los registros de EF Core desde cualquier tipo de aplicación mediante el uso de LogTo al configurar una instancia de DbContext.

```
protected override void  
OnConfiguring(DbContextOptionsBuilder optionsBuilder)  
=> optionsBuilder.LogTo(Console.WriteLine);
```

© JMA 2016. All rights reserved

## DATOS

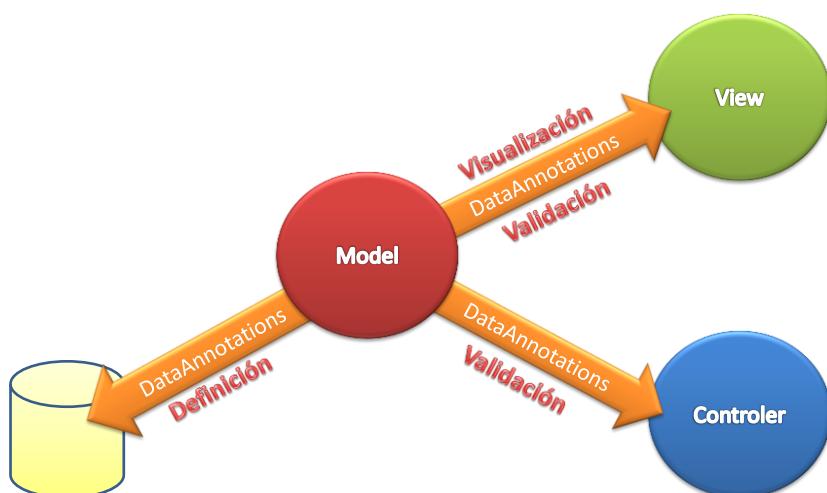
© JMA 2016. All rights reserved

## DataAnnotations

- System.ComponentModel.DataAnnotations;
- Permiten definir los metadatos de los modelos de datos para su uso por entidades externas.
- Principio DRY: No repetir
- Cuenta con decoradores para:
  - Definición del modelo de datos: claves (PK), asociaciones (FK), simultaneidad, ...
  - Interfaz de usuario y localización: Título, descripción, formato, solo lectura, ...
  - Validaciones y errores personalizados: Obligatoriedad, longitudes, formatos, ...

© JMA 2016. All rights reserved

## Contexto Declarativo



© JMA 2016. All rights reserved

# Visualización

Decorador	Descripción
DataType	Especifica el nombre de un tipo adicional que debe asociarse a un campo de datos.
Display	Permite especificar las cadenas traducibles de los tipos y miembros de las clases parciales de entidad.
DisplayFormat	Especifica el modo en que los datos dinámicos de ASP.NET muestran y dan formato a los campos de datos.
ScaffoldColumn	Especifica si una clase o columna de datos usa la técnica scaffolding.
ScaffoldTable	Especifica si una clase o tabla de datos usa la técnica scaffolding.
UIHint	Especifica la plantilla o el control de usuario que los datos dinámicos usan para mostrar un campo de datos.
HiddenInput	(Microsoft.AspNetCore.Mvc) Indica que una propiedad se debería presentar como un elemento input oculto.

© JMA 2016. All rights reserved

## [Display]

- **Name:** valor que se usa para mostrarlo en la interfaz de usuario (Título, Etiqueta, ...).
- **Description:** valor que se usa para mostrar una descripción en la interfaz de usuario.
- **Prompt:** valor que se usará para establecer la marca de agua para los avisos en la interfaz de usuario.
- **ShortName:** valor que se usa para la etiqueta de columna de la cuadrícula.
- **GroupName:** valor que se usa para agrupar campos en la interfaz de usuario.
- **Order:** número del orden de la columna.

© JMA 2016. All rights reserved

## Tipos asociados

Asociado	Descripción
DateTime	Representa un instante de tiempo, expresado en forma de fecha y hora del día.
Date	Representa un valor de fecha.
Time	Representa un valor de hora.
Duration	Representa una cantidad de tiempo continua durante la que existe un objeto.
PhoneNumber	Representa un valor de número de teléfono.
Currency	Representa un valor de divisa.
Text	Representa texto que se muestra.
Html	Representa un archivo HTML.
MultilineText	Representa texto multilínea.
EmailAddress	Representa una dirección de correo electrónico.
Password	Representa un valor de contraseña.
Url	Representa un valor de dirección URL.
ImageUrl	Representa una URL en una imagen.
CreditCard	Representa un número de tarjeta de crédito.
PostalCode	Representa un código postal.
Upload	Representa el tipo de datos de la carga de archivos.

© JMA 2016. All rights reserved

## Validación

Decorador	Descripción
Required	Especifica que un campo de datos necesita un valor.
DataType	Especifica el nombre de un tipo adicional que debe asociarse a un campo de datos. Decoradores especializados: CreditCard, EmailAddress, EnumDataType, FileExtensions, Phone, Url
Compare	Compara dos propiedades.
Range	Especifica las restricciones de intervalo numérico para el valor de un campo de datos.
StringLength	Especifica la longitud mínima y máxima de caracteres que se permiten en un campo de datos. Decoradores especializados: MinLength, MaxLength
RegularExpression	Especifica que un valor de campo de datos debe coincidir con la expresión regular especificada.
CustomValidation	Especifica un método de validación personalizado que se utiliza para validar una propiedad o una instancia de clase.

© JMA 2016. All rights reserved

# Validación

Decorador	Descripción
CreditCard	Especifica que el valor de un campo de datos es un número de tarjeta de crédito. Requiere métodos adicionales de validación de jQuery.
EmailAddress	Valida que la propiedad tiene un formato de correo electrónico.
Phone	Especifica que un valor de campo de datos es un número de teléfono con formato correcto.
Url	Valida que la propiedad tiene un formato de dirección URL.
Remote	Valida la entrada en el cliente mediante una llamada a un método de acción en el servidor. Está en el espacio de nombres Microsoft.AspNetCore.Mvc.
ValidateNever	Indica que una propiedad o parámetro debe excluirse de la validación. Está en el espacio de nombres Microsoft.AspNetCore.Mvc.ModelBinding.Validation.

© JMA 2016. All rights reserved

## Validación manual de objetos

- En System.ComponentModel.DataAnnotations, la clase estática Validator ofrece métodos que permiten realizar las comprobaciones de forma directa sobre objetos o propiedades concretas.

```
IEnumerable<ValidationResult> getValidationErrors(object obj) {
    var validationResults = new List<ValidationResult>();
    var context = new ValidationContext(obj, null, null);
    Validator.TryValidateObject(obj,
        context,
        validationResults,
        true);
    return validationResults;
}
```

© JMA 2016. All rights reserved

## IValidableObject

- Obliga a implementar un único método, llamado Validate(), que determina si el objeto especificado es válido.
  - Será invocado automáticamente por TryValidateObject() siempre que no encuentre errores al comprobar las restricciones especificadas mediante anotaciones.
  - Devolverá una lista de objetos ValidationResult con los resultados de las comprobaciones.
  - En las clases prescriptivas (EF) se aplica con una partial class.
- ```
public partial class Persona : IValidableObject {
    public IEnumerable<ValidationResult> Validate(ValidationContext validationContext) {
        if (FechaNacimiento.Date.CompareTo(DateTime.Today) > 0)
            yield return new ValidationResult("Todavía no ha nacido", new[]
                { nameof(FechaNacimiento) });
    }
}
```
- Interfaces relacionados: IDataErrorInfo, INotifyDataErrorInfo

© JMA 2016. All rights reserved

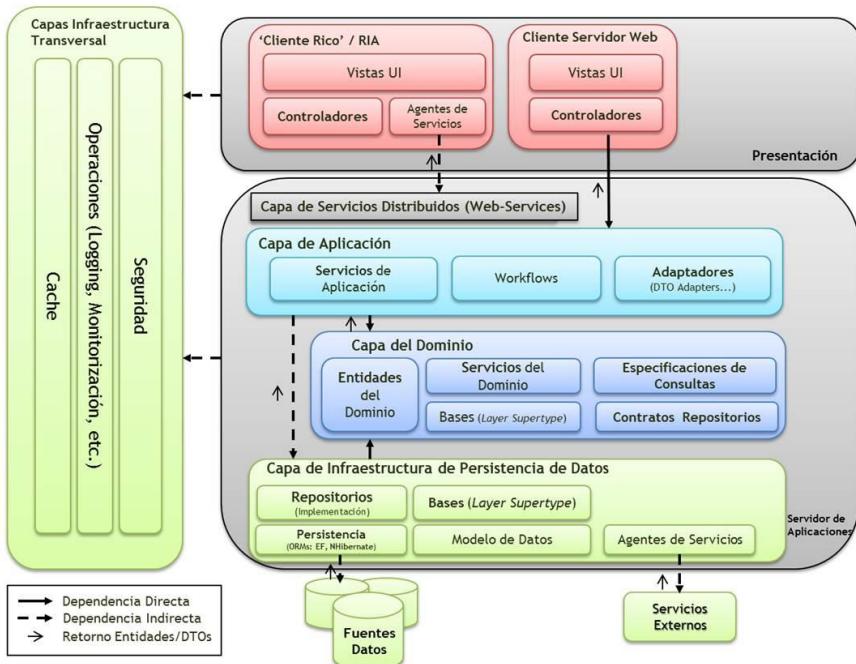
## Anotar clases ya existentes

- Se requiere una clase auxiliar con las propiedades a decorar decoradas.
- Declarativa: [MetadataType(typeof(tipo))]  
– Se aplica con una partial class (en el mismo ensamblado que la clase a anotar).
- Imperativa: System.ComponentModel.TypeDescriptor y AssociatedMetadataTypeTypeDescriptionProvider:  

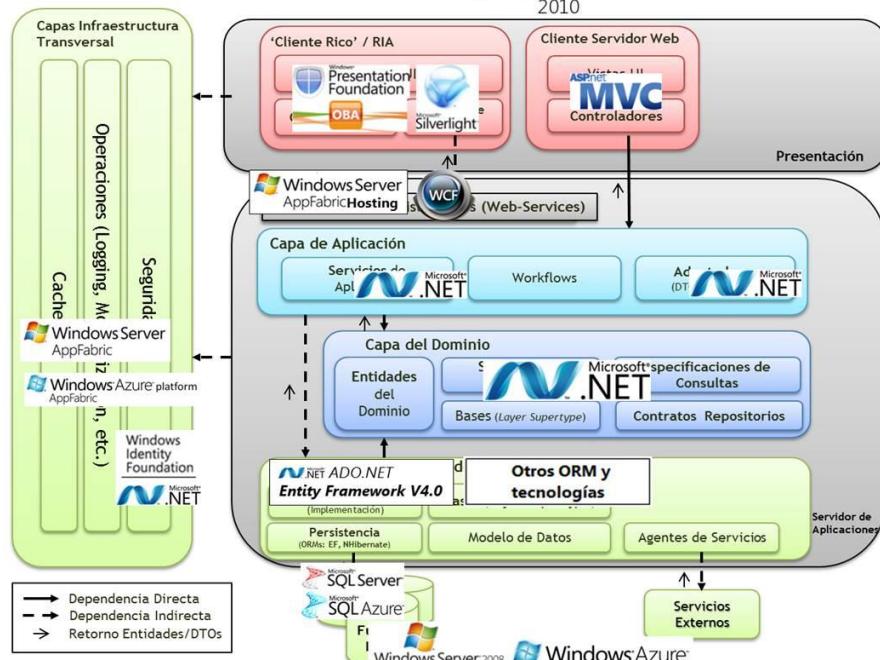
```
var descriptionProvider = new
    AssociatedMetadataTypeTypeDescriptionProvider(
        typeof(Friend), typeof(FriendMetadata));
TypeDescriptor.AddProviderTransparent(
    descriptionProvider, typeof(Friend));
```

© JMA 2016. All rights reserved

## Arquitectura N-Capas con Orientación al Dominio



Mapeo de Tecnologías 'Wave .NET 4.0'



# Repository

- Un repositorio separa la lógica empresarial de las interacciones con la base de datos subyacente y centra el acceso a datos en un área, lo que facilita su creación y mantenimiento.
- El repositorio pertenecen a la capa de infraestructura y devuelve los objetos del modelo de dominio.
- Deberían implementar el patrón de doble herencia.
- Forma parte de los Domain Driven Design patterns: Domain Entity, Value-Object, Aggregates, Repository, Unit of Work, Specification, Dependency Injection, Inversion of Control (IoC).

© JMA 2016. All rights reserved

## Ventajas del Repositorio

- Proporciona un punto de sustitución para las pruebas unitarias. Es fácil probar la lógica empresarial sin una base de datos y otras dependencias externas.
- Las consultas y los modelos de acceso a datos duplicados se pueden quitar y refactorizar en el repositorio.
- Los métodos del controlador pueden usar parámetros fuertemente tipados, lo que significa que el compilador encontrará errores de tipado de datos en cada compilación en lugar de realizar la búsqueda de errores de tipado de datos en tiempo de ejecución durante las pruebas.
- El acceso a datos está centralizado, lo que brinda las siguientes ventajas:
  - Mayor separación de intereses (SoC), uno de los principios de MVC, lo que facilita más el mantenimiento y la legibilidad.
  - Implementación simplificada de un almacenamiento en caché de datos centralizado.
  - Arquitectura más flexible y con menor acoplamiento, que se puede adaptar a medida que el diseño global de la aplicación evoluciona.
- El comportamiento se puede asociar a los datos relacionados (calcular campos, aplicar relaciones, reglas de negocios complejas entre los elementos de datos de una entidad, ...).
- Un modelo de dominio se puede aplicar para simplificar una lógica empresarial compleja.

© JMA 2016. All rights reserved

## Ampliación

- Domain Oriented N-Layered .NET 4.0 Sample App
  - <http://microsoftnlayerapp.codeplex.com/>

© JMA 2016. All rights reserved

## ENTRADA Y SALIDA

© JMA 2016. All rights reserved

# Archivos y directorios

## System.IO

- Estas son algunas clases de archivo y directorio de uso general:
  - File: proporciona métodos estáticos para crear, copiar, eliminar, mover y abrir archivos, y ayuda a crear un objeto FileStream.
  - FileInfo: proporciona métodos de instancia para crear, copiar, eliminar, mover y abrir archivos, y ayuda a crear un objeto FileStream.
  - Directory: proporciona métodos estáticos para crear, mover y enumerar directorios y subdirectorios.
  - DirectoryInfo: proporciona métodos de instancia para crear, mover y enumerar directorios y subdirectorios.
  - Path: proporciona métodos y propiedades para procesar cadenas de directorio entre plataformas.

© JMA 2016. All rights reserved

# Secuencias

- Las secuencias comprenden tres operaciones fundamentales:
  - Lectura: transferencia de datos desde una secuencia a una estructura de datos como, por ejemplo, una matriz de bytes.
  - Escritura: transferencia de datos a una secuencia desde un origen de datos.
  - Búsqueda: consulta y modificación de la posición actual en una secuencia.
- Estas son algunas de las clases de secuencias de uso general:
  - FileStream: para leer y escribir en un archivo.
  - IsolatedStorageFileStream: para leer y escribir en un archivo en almacenamiento aislado.
  - MemoryStream: para leer y escribir en la memoria como una memoria auxiliar.
  - BufferedStream: para mejorar el rendimiento de las operaciones de lectura y escritura.
  - NetworkStream: para leer y escribir sobre los sockets de red.
  - PipeStream: para leer y escribir sobre canalizaciones anónimas y con nombre.
  - CryptoStream: para vincular secuencias de datos con transformaciones criptográficas.

© JMA 2016. All rights reserved

## Lectores y escritores

- Estas son algunas clases de lectura y escritura de uso general:
  - BinaryReader y BinaryWriter: para leer y escribir tipos de datos primitivos como valores binarios.
  - StreamReader y StreamWriter: para leer y escribir caracteres utilizando un valor de codificación para convertir los caracteres en bytes y a la inversa.
  - StringReader y StringWriter: para leer y escribir caracteres en cadenas.
  - TextReader y TextWriter: sirven como clases base abstractas para otros lectores y escritores que leen y escriben caracteres y cadenas, pero no datos binarios

© JMA 2016. All rights reserved

## Compresión

- Las clases siguientes se utilizan con frecuencia al comprimir y descomprimir archivos y secuencias:
  - ZipArchive: para crear y recuperar entradas en el archivo zip.
  - ZipArchiveEntry: para representar un archivo comprimido.
  - ZipFile: para crear, extraer y abrir un paquete comprimido.
  - ZipFileExtensions: para crear y extraer entradas en un paquete comprimido.
  - DeflateStream: para comprimir y descomprimir secuencias utilizando el algoritmo de deflación (Deflate).
  - GZipStream: para comprimir y descomprimir secuencias con formato de datos gzip.

© JMA 2016. All rights reserved

# Serialización

- System.Runtime.Serialization
- Atributos: SerializableAttribute, NonSerializedAttribute.
- Serialización
  - Binaria: Formatter.
  - XML: XmlSerializer
  - SOAP: WCF
  - JSON: NuGet
  - Personalizada: ISerializable

© JMA 2016. All rights reserved

# PROGRAMACIÓN CONCURRENTE

© JMA 2016. All rights reserved

# Introducción

- Cuando:
  - Se dependa de recursos muy lentos (Web, disco, red, servidores, ...)
  - Se requiera simultaneidad (animaciones, juegos, ...)
  - Se disponga de múltiples procesadores/cores.
- Opciones:
  - Programación asíncrona
    - Componente BackgroundWorker
  - Programación paralela
    - Subprocesamiento administrado
    - Task Parallel Library, biblioteca de procesamiento paralelo basado en tareas (TPL)
    - Parallel LINQ, implementación paralela de LINQ to Objects (PLINQ).

© JMA 2016. All rights reserved

## Patrones para la programación asíncrona

- Patrón asíncrono basado en tareas (TAP) , que utiliza un método único para representar el inicio y la finalización de una operación asíncrona. TAP se presentó en .NET Framework 4. Es el enfoque recomendado para la programación asíncrona en .NET. Las palabras clave `async` y `await` en C# y los operadores `Async` y `Await` en Visual Basic agregan compatibilidad de lenguaje para TAP.
- El modelo asíncrono basado en eventos (EAP), que es el patrón heredado basado en eventos para proporcionar el comportamiento asíncrono. Requiere un método con el sufijo `Async`, así como uno o más eventos, tipos de delegado de controlador de eventos y tipos derivados de `EventArgs`. EAP se presentó en .NET Framework 2.0. Ya no se recomienda para nuevo desarrollo.
- El patrón Modelo de programación asíncrona (APM) (también denominado `IAsyncResult`), que es el modelo heredado que usa la interfaz `IAsyncResult` para ofrecer un comportamiento asíncrono. En este patrón, las operaciones sincrónicas requieren los métodos `Begin` y `End` (por ejemplo, `BeginWrite` y `EndWrite` para implementar una operación de escritura asíncrona). Este patrón ya no se recomienda para nuevo desarrollo.

© JMA 2016. All rights reserved

## Modelos de diseño asíncronos

- Una operación asíncrona se ejecuta en un subproceso independiente del subproceso de aplicación principal. Cuando una aplicación llama a métodos para que realicen una operación de forma asíncrona, la aplicación puede seguir ejecutándose mientras el método asíncrono efectúa su tarea.
- Programación asíncrona mediante `IAsyncResult`
  - Métodos:
    - Método Síncrono: `NombreDeOperación`
    - Métodos Asíncronos: `BeginNombreDeOperación` y `EndNombreDeOperación`
- Programación asíncrona mediante delegados
  - Métodos:
    - Método Síncrono: `Invoke`
    - Métodos Asíncronos: `BeginInvoke` y `EndInvoke`
- Programación asíncrona basada en eventos
  - Métodos:
    - Método Síncrono: `NombreDeMétodo`
    - Métodos Asíncronos: `NombreDeMétodoAsync`, `NombreDeMétodoAsyncCancel` (o `CancelAsync`)
  - Evento: `NombreDeMétodoCompleted`

© JMA 2016. All rights reserved

## Componente BackgroundWorker

- El componente `BackgroundWorker` ofrece la posibilidad de ejecutar operaciones prolongadas de forma asíncrona ("en segundo plano"), en un subproceso diferente del subproceso principal de la interfaz de usuario de la aplicación.
- Para usar un `BackgroundWorker`, solo tiene que decirle qué método de trabajo prolongado debe ejecutar en segundo plano y, después, llame al método `RunWorkerAsync`.
- El subproceso que realiza la llamada continúa ejecutándose normalmente mientras el método de trabajo se ejecuta asíncronamente. Cuando el método termina, el `BackgroundWorker` alerta al subproceso que realizó la llamada activando el evento `RunWorkerCompleted` que, opcionalmente, contiene el resultado de la operación.

© JMA 2016. All rights reserved

# Subprocesamiento administrado

- Crear subproceso  
Thread hilo = new Thread(new ThreadStart(Operacion));
- Lanzar subproceso  
hilo.Start();
- Abortar subproceso  
hilo.Abort();
- Bloquear el subproceso de llamada hasta que termina el subproceso.  
hilo.Join();
- Bloquear el subproceso durante el número de milisegundos especificado.  
Thread.Sleep(0);
- Bloquear subprocesos por bloqueo de recursos.  
lock(recurso) {  
...  
}

© JMA 2016. All rights reserved

## Sincronización de subprocessos y contención

| Clase                   | Descripción                                                                                                                                                                                 |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Mutex</b>            | Primitiva de sincronización que puede utilizarse también para la sincronización entre procesos.                                                                                             |
| <b>Monitor</b>          | Proporciona un mecanismo que sincroniza el acceso a los objetos.                                                                                                                            |
| <b>Interlocked</b>      | Proporciona operaciones atómicas para las variables compartidas por varios subprocessos.                                                                                                    |
| <b>ReaderWriterLock</b> | Define un bloqueo que admite un escritor y varios lectores.                                                                                                                                 |
| <b>Semaphore</b>        | LIMITA el número de subprocessos que pueden tener acceso a un recurso o grupo de recursos simultáneamente.                                                                                  |
| <b>ThreadPool</b>       | Proporciona un grupo de subprocessos que pueden utilizarse para exponer elementos de trabajo, procesar la E/S asíncrona, esperar en nombre de otros subprocessos y procesar temporizadores. |

© JMA 2016. All rights reserved

# Modelo asincrónico basado en tareas (TAP)

- El patrón asincrónico basado en tareas (TAP) se basa en los tipos `System.Threading.Tasks.Task` y `System.Threading.Tasks.Task<TResult>` del espacio de nombres `System.Threading.Tasks`, que se usan para representar operaciones asincrónicas arbitrarias. TAP es el patrón asincrónico de diseño recomendado para los nuevos desarrollos.
- TAP usa un solo método para representar el inicio y la finalización de una operación asincrónica.
- Un método asincrónico basado en TAP puede hacer una pequeña cantidad de trabajo sincrónicamente, como validar argumentos e iniciar la operación asincrónica, antes de que devuelva la tarea resultante. El trabajo sincrónico debe reducirse al mínimo de modo que el método asincrónico pueda volver rápidamente.
- A partir de .NET Framework 4.5, cualquier método que tenga la palabra clave `async` se considera un método asincrónico, y los compiladores realizan las transformaciones necesarias para implementar el método de forma asincrónica mediante TAP. Un método asincrónico debe devolver un objeto `System.Threading.Tasks.Task` o `System.Threading.Tasks.Task<TResult>`.

© JMA 2016. All rights reserved

## Tareas

- Crear una tarea:  

```
Action<object> action = (object obj) => {
    // ...
};

Task t1 = new Task(action, "alpha");
```
- Lanzar una tarea:  

```
t1.Start();
```
- Esperar a que se termine la tarea:  

```
t1.Wait();
```
- Crear y ejecutar una tarea:  

```
Task t1 = Task.Run(() => {
    // ...
});
```
- Crear y ejecutar una tarea que devuelve un valor:  

```
var t1 = Task<int>.Run(() => {
    // ...
    return value;
});

Console.WriteLine("Finished with {0}: {1}.", t1.Result);
```

© JMA 2016. All rights reserved

## Métodos asíncronos

- Los parámetros de un método de TAP deben coincidir con los parámetros de su homólogo sincrónico y se deben proporcionar en el mismo orden. Sin embargo, los parámetros out y ref están exentos de esta regla y se deben evitar completamente. En su lugar, los datos que se hubieran devuelto con return o con un parámetro out o ref se deben devolver como parte del tipo TResult devuelto por Task< TResult >.
- En TAP, la cancelación es opcional tanto para los implementadores de método asíncrono como para los consumidores de este método. Si una operación permite la cancelación, expone una sobrecarga del método asíncrono que acepta un token de cancelación (instancia de CancellationToken). Por convención, el parámetro se denomina cancellationToken.
- En TAP, el progreso se controla a través de una interfaz IProgress< T >, la cual se pasa al método asíncrono como un parámetro normalmente denominado progress.
- Las sobrecargas que se suelen proporcionar son:
 

```
public Task MethodNameAsync(...);
public Task MethodNameAsync(..., CancellationToken cancellationToken);
public Task MethodNameAsync(..., IProgress<T> progress);
public Task MethodNameAsync(..., CancellationToken cancellationToken, IProgress<T> progress);
```

© JMA 2016. All rights reserved

## Métodos asíncronicos (ver. 5.0)

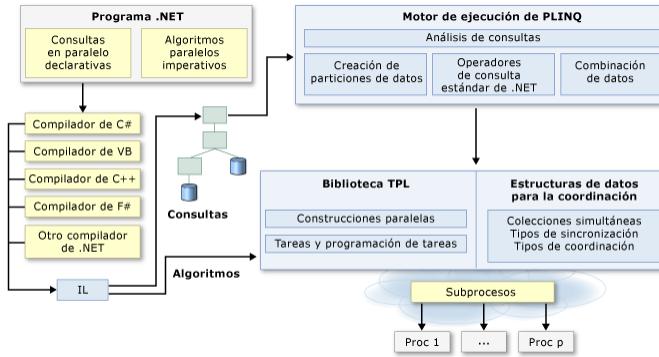
- Las palabras clave **async** y **await** permiten crear un método asíncrono casi tan fácilmente como se crea un método sincrónico.

```
async string MetodoAsync() {
    Task<string> tarea = ...;
    // ...
    string rslt = await tarea;
    // ...
    return rslt;
}
```

© JMA 2016. All rights reserved

# Biblioteca de procesamiento paralelo basado en tareas (TPL)

- La biblioteca TPL (Task Parallel Library, biblioteca de procesamiento paralelo basado en tareas) es un conjunto de API y tipos públicos de los espacios de nombres System.Threading y System.Threading.Tasks.



© JMA 2016. All rights reserved

## Clase Parallel

- Proporciona compatibilidad con regiones y bucles paralelos.
- Para ejecutar un bucle for en el que es posible ejecutar iteraciones en paralelo:
 

```
Parallel.For(0, N, i => {
    //...
});
```
- Para ejecutar un bucle foreach, si no interfiere el paralelismo:
 

```
Parallel.ForEach(list, item => {
    // ...
});
```
- Para ejecutar cada una de las acciones proporcionadas, posiblemente en paralelo.
 

```
Parallel.Invoke(action1, action2, () => {
    //...
});
```

© JMA 2016. All rights reserved

## Parallel LINQ (PLINQ)

- Parallel LINQ (PLINQ) es una implementación en paralelo del patrón Language-Integrated Query (LINQ). PLINQ implementa el conjunto completo de operadores de consulta estándar de LINQ como métodos de extensión para el espacio de nombres System.Linq y tiene operadores adicionales para las operaciones en paralelo. PLINQ combina la simplicidad y legibilidad de la sintaxis de LINQ con la eficacia de la programación en paralelo.

```
var source = Enumerable.Range(1, 10000);
var evenNums = from num in source.AsParallel()
               where num % 2 == 0
               select num;
Console.WriteLine("{0} even numbers out of {1} total",
    evenNums.Count(), source.Count());
```

© JMA 2016. All rights reserved

Aplicaciones de escritorio

## INTRODUCCIÓN A WINDOWS FORMS

© JMA 2016. All rights reserved

## ¿Qué es Windows Forms?

- Windows Forms es un subconjunto de la .NET Framework Class Library que permite el desarrollo de aplicaciones de escritorio ricas bajo Microsoft Windows.
- Incluye clases base, interfaces, enumeraciones y controles gráficos diversos.
- El framework .NET incluye una gran cantidad de clases para la creación y administración de aplicaciones Windows, y como con el resto de los objetos de .NET, los objetos de Windows Forms pueden ser heredados y de esa manera extender su funcionalidad.
- Si no es suficiente con los controles que expone el NameSpace System.Windows.Forms y los NameSpaces asociados, es posible crear nuestras propias clases que hereden de algunas de la propuestas por las librerías de clases base y así lograr la funcionalidad requerida.

© JMA 2016. All rights reserved

## ¿Qué es un formulario?

- Un formulario Windows Forms actúa como interfaz del usuario local de Windows.
- Los formularios pueden ser ventanas estándar, interfaces de múltiples documentos (MDI), cuadros de diálogo, etc.
- Los formularios son clases que exponen propiedades, métodos que definen su comportamiento y eventos que definen la interacción con el usuario.

© JMA 2016. All rights reserved

## El diseñador de formularios

- Al momento de diseñar un formulario, el diseñador de Visual Studio escribe de forma automática el código que describe a cada uno de los controles y al propio formulario.
- El concepto de Partial class que incorpora .NET 2.0 permite separar el código de una clase en varios archivos fuentes diferentes.
- El diseñador de formularios utiliza esta técnica para escribir en un archivo aparte todo el código que él mismo genera.
- Esto permite organizar más claramente el código, manteniendo separada la lógica de la aplicación en un archivo diferente.

© JMA 2016. All rights reserved

## Generalidades

- El objeto Form es el principal componente de una aplicación Windows.  
`Form miForm = new Form();`
- Algunas de sus propiedades admiten valores de alguno de los tipos nativos de .NET  
`miForm.ShowInTaskBar = false;`  
`miForm.Opacity = 0.83;`
- Otras propiedades requieren la asignación de objetos:  
`miForm.Size = new Size(100, 100);`  
`miForm.Location = new Location(0, 0);`

© JMA 2016. All rights reserved

# Métodos

- **Show()**
  - Visualiza el formulario. Puede especificarse su formulario Owner.
    - Si un formulario A es owner (dueño) de otro B, el formulario B siempre se visualizará sobre el A, sin importar si otro formulario está activo.
  
- **ShowDialog()**
  - Visualiza el formulario como cuadro de diálogo Modal.
    - Un formulario visualizado de forma modal no permite que otro formulario perteneciente a la misma aplicación tome foco. Esta opción es utilizada para mostrar cuadros de diálogo y focalizar la atención del usuario.

© JMA 2016. All rights reserved

# Eventos

- **Manejadores de eventos**
    - Por cada evento soportado por el Form (o por cualquier otro objeto) es posible definir varios métodos controladores.
    - A su vez, un método manejador puede controlar eventos disparados por diferentes objetos.
- ```
// Varios manejadores para un evento
this.Click += new EventHandler(MetodoManejador1);
this.Click += new EventHandler(MetodoManejador2);
// Un mismo manejador para diferentes eventos
this.Load += new EventHandler(ManejadorCentralizado);
this.Activated +=new EventHandler(ManejadorCentralizado);
```

© JMA 2016. All rights reserved

## Ciclo de vida del formulario

- Muchos de los eventos a los que responde el objeto Form pertenecen al ciclo de vida del formulario
- Entre estos eventos se encuentran los siguientes, en orden de ocurrencia:
  - Load: El formulario está en memoria, pero invisible.
  - Paint: Se “pinta” el formulario y sus controles.
  - Activated: El formulario recibe foco.
  - FormClosing: Se va a cerrar, permite cancelar el cierre.
  - FormClosed: El formulario es invisible.
  - Disposed: El objeto está siendo destruido.

© JMA 2016. All rights reserved

## Trabajando con el Mouse

- El mouse puede ser controlado escribiendo código para alguno de estos eventos:
  - MouseClick
  - MouseEnter
  - MouseMove
- A través de los argumentos que reciben los manejadores de estos eventos se puede obtener:
  - La posición del puntero
  - Qué botón fue presionado
  - Cantidad de “pasos” que fue girada la rueda

© JMA 2016. All rights reserved

## Trabajando con el Teclado

- El manejador del evento KeyPress informa a través del argumento e.KeyChar el código de la tecla presionada.
- Es posible cancelar el comportamiento por defecto asignando “true” al argumento e.Handled.
- Los argumentos que reciben los manejadores de los eventos KeyDown y KeyUp informan del estado de las teclas Alt, Ctrl y Shift.
- El evento HelpRequested es disparado cuando se presiona la tecla F1.

© JMA 2016. All rights reserved

## Foco y orden de tabulación

- El objeto Form expone diferentes propiedades, métodos y eventos que permiten controlar la navegabilidad del formulario:
  - Propiedad CanFocus: Indica si el control puede tomar foco.
  - Propiedad Focused: Indica si el control tiene el foco actualmente.
  - Método Focus(): “Mueve” el foco al objeto deseado.
- Orden de tabulación (Propiedad TabIndex)
  - En forma visual, desde el diseñador de formularios, es posible configurar el orden en el que el foco se irá moviendo por los controles.

© JMA 2016. All rights reserved

## MessageBox

- Para mostrar información o pedir intervención del usuario, es posible utilizar la clase MessageBox.
- Esta clase contiene métodos estáticos que permiten mostrar un cuadro de mensaje para interactuar con el usuario de la aplicación.
- Los parámetros se especifican a través de enumerados que facilitan la legibilidad del código, por ejemplo:
  - MessageBoxButtons.AbortRetryIgnore
  - MessageBoxIcon.Error
  - MessageBoxDefaultButton.Button1

© JMA 2016. All rights reserved

## Controles de Windows (1/3)

- Gran parte del éxito de una aplicación Windows consiste en elegir y manejar adecuadamente los controles que ofrece .NET.
- Entre los controles nativos se encuentran controles totalmente nuevos y versiones mejoradas de sus pares de .NET 1.1.
- Nuevos controles como el control BindingSource mejoran notablemente el enlace de datos provenientes de muy diferentes fuentes de datos.

© JMA 2016. All rights reserved

## Controles de Windows (2/3)

- **MaskedEdit**
  - Es un control que permite el uso de máscaras personalizadas para facilitar la entrada de datos.
- **TextBox**
  - Cuadro de texto que, entre otras mejoras tiene la funcionalidad de auto completar.
- **Label**
  - Si el texto ocupa más lugar que el largo del control, gracias a la nueva propiedad AutoEllipsis incorporada en .NET 2.0, el excedente se reemplaza automáticamente con tres puntos (...)



© JMA 2016. All rights reserved

## Controles de Windows (3/3)

- **DataGridView**
  - Es una versión mejorada del DataGrid control de .NET 1.1 con funcionalidad de modo “Virtual”. Permite enlazar datos originados en una Base de Datos a medida que se necesitan.
- **TreeView**
  - Utilizando la nueva propiedad DrawMode es posible sobreescribir la manera en que el sistema operativo “dibuja” cada nodo del árbol.

Surname	Given Name	Date of Birth	Age	Gender	Marital Status	Salaried
Goldberg	Joséef	February, 1998	56	M	S	✓
Duffy	Teni	March, 1998	44	F	S	✓
Higa	Sidney	March, 1998	59	M	M	□
Maxwell	Taylor	March, 1998	59	M	M	□
Ford	Jeffrey	March, 1998	59	M	S	□
Brown	Jim	March, 1998	59	M	S	□
Hanson	Doto	April, 1998	59	F	M	✓
Guenther	John	April, 1998	59	M	M	□
Glemp	Diane	April, 1998	59	F	M	□
Johnson	Barry	February, 1998	59	M	S	□
Krebs	Pete	January, 1998	32	M	M	✓
Erickson	Gail	February, 1998	63	F	M	✓
Eberbrock	Ruth	February, 1998	59	F	M	□
Dobney	JoLynn	January, 1998	59	F	S	□
Mu	Zheng	January, 1999	31	M	S	□
Bradley	David	January, 1998	40	M	S	✓
Komornitski	Paul	January, 1999	34	M	S	□

© JMA 2016. All rights reserved

## Controles Contenedores

- Algunos controles como el propio Form, Panel o GroupBox heredan de la clase ContainerControl en lugar de hacerlo directamente de Control.
- Por este motivo, poseen una colección mediante la que se puede acceder a los controles que contiene.
- Sólo se puede acceder a los controles de nivel superior, no a todos los controles contenidos.

© JMA 2016. All rights reserved

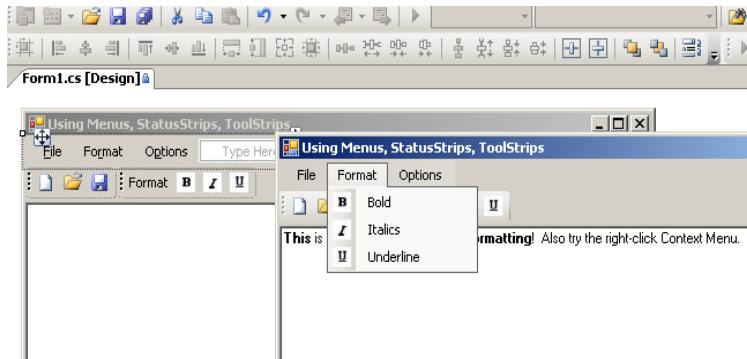
## Menú (1/2)

- El nuevo control MenuStrip provee un sistema de menú para un formulario.
- MenuStrip es contenedor de objetos como ToolStripMenuItem, ToolStripComboBox, ToolStripSeparator, ToolStripTextBox.
- El control ContextMenuStrip representa un menú que será mostrado al usuario cuando presione el botón derecho del mouse. También puede contener los mismos controles que MenuStrip.
- Las propiedades MergeAction y MergeIndex del objeto ToolStripItem permiten controlar la manera en que los menú de dos diferentes ventanas se “mezclarán”.

© JMA 2016. All rights reserved

## Menú (2/2)

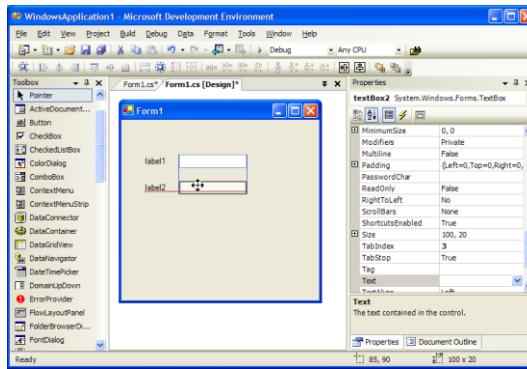
- En la imagen se ve una aplicación que utiliza los controles MenuStrip y ToolStrip. En segundo plano se ve el diseñador de formularios.



© JMA 2016. All rights reserved.

## Diseño del Interfaz: Snaplines

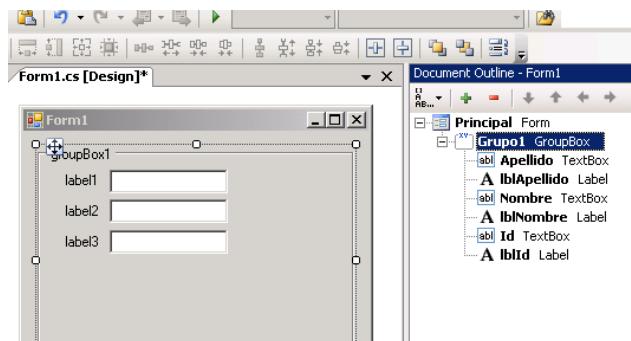
- Son líneas que se dibujan automáticamente en el diseñador de formularios al momento de posicionar el control.
- Ayudan a mantener la correcta distancia entre los controles y entre éstos y su contenedor.



© JMA 2016. All rights reserved.

## Diseño del Interfaz: Document Outline

- Mediante esta herramienta es posible ver la jerarquía de controles del formulario en forma de TreeView, y además editar el nombre de esos controles.



© JMA 2016. All rights reserved

## Diseño del Interfaz: Layout Panels

- **TableLayoutPanel**
  - Es similar a diseñar una tabla en un formulario HTML.
  - Facilita la ubicación de los controles en escenarios de localización.
  - Facilita la creación de interfaces redimensionables.
- **FlowLayoutPanel**
  - Los controles contenidos “fluyen” como en el modo por defecto de un formulario HTML.

© JMA 2016. All rights reserved

## Diseño del Interfaz: Anchor y Docking

- **Anchor**
  - Automatiza el redimensionamiento y posicionamiento de los controles cuando se redimensiona el formulario.
  - Los controles pueden anclarse contra cualquier combinación de los cuatro bordes del formulario.
- **Docking**
  - La propiedad Dock (que exponen todos los controles Windows) permite pegar un control a alguno de los cuatro bordes del formulario.

© JMA 2016. All rights reserved

## Controles Extender Providers

- Son controles que, una vez colocados en un formulario, agregan nuevas propiedades a los otros controles existentes.
  - ErrorProvider: Permite asociar un error a un control mostrando un ícono que parpadea al lado de dicho control.
  - HelpProvider: Permite asociar a un control desde una simple cadena de texto un archivo Help que serán mostrados al presionar F1.
  - ToolTip: Es el clásico rectángulo que aparece asociado a un control y que es mostrado cuando el mouse se detiene sobre él.

© JMA 2016. All rights reserved

## Herencia Visual

- Dado que un formulario Windows es como cualquier otra clase .NET, es posible aplicar herencia.
- Al heredar de un formulario base, además de sus miembros, se heredan todos los controles que en él se encuentren.
- La herencia visual es una poderosa herramienta que permite estandarizar el diseño y el comportamiento de los formularios a lo largo del proyecto, o inclusive, en diferentes proyectos.
- Permite entre otras cosas:
  - Unificar el diseño de las interfaces de usuario.
  - Reutilizar funcionalidad de formularios similares.

© JMA 2016. All rights reserved

## Configuración

- Las Propiedades Dinámicas permiten almacenar preferencias del usuario en archivos de configuración asociados a la aplicación.
- Estos valores pueden ser leídos y grabados tanto en diseño como en ejecución.
- Por cada valor que se almacena se puede definir el nombre, tipo de dato y alcance (usuario o aplicación).
- Es posible además enlazar (binding) propiedades dinámicas a controles del formulario.

© JMA 2016. All rights reserved

## Diálogos Comunes

- Los cuadros de diálogo comunes permiten interacción con el usuario para ejecutar acciones comunes como abrir un archivo, configurar la impresión, seleccionar un color del sistema, etc.
- Sólo basta configurar algunas propiedades e invocar su método ShowDialog().
- Alguno de los controles que muestran estos diálogos son:
  - ColorDialog
  - PrintDialog
  - SaveDialog
  - OpenFileDialog

© JMA 2016. All rights reserved

## Objeto BindingSource

- El objeto BindingSource permite el enlace de controles a datos provenientes de fuentes de datos (DataSource) de tres tipos
  - DataBase: Crea internamente un dataset.
  - WebService: Crea una referencia web a un servicio que es el que proporciona los datos
  - Object: Utiliza una clase de negocios como fuente de datos creando automáticamente una colección de elementos de esa clase.
- Usándolo junto a un control DataBindingNavigator y un DataGridView conforman un formulario de ABM sin escribir código alguno.

© JMA 2016. All rights reserved

## Cuadros de diálogo

- FormBorderStyle = FixedDialog
- ControlBox, MinimizeBox y MaximizeBox en false
- dlg1.ShowDialog()
- DialogResult
- parentForm
- MessageBox.show

© JMA 2016. All rights reserved

## Formularios MDI

- Formulario Padre:
  - IsMdiContainer
  - MdiChildren
  - ActiveMdiChild
  - LayoutMdi: MdiLayout  
(Arrangelcons,Cascade,TileHorizontal,TileVertical) Mosaico
- Evento: MdiChildActivate
- En el menú: MDIList
- Formularios Hijos:
  - IsMdiChild
  - MdiParent
  - Diferencias con owner y ownedForms

© JMA 2016. All rights reserved

Aplicaciones de escritorio

## WINDOWS PRESENTATION FOUNDATION

© JMA 2016. All rights reserved

### ¿Qué es WPF?

- WPF es una tecnología para desarrollar la siguiente generación de aplicaciones en Windows y en la web, utilizando toda la potencia del hardware y creando las mejores experiencias de usuario (UX).

© JMA 2016. All rights reserved

# Windows Presentation Foundation

- **Tecnología IU estratégica de Microsoft**
  - Plataforma y motor
  - Lo mejor del Web y Windows
  - Uso de GPU para alto rendimiento
- **Unificación**
  - Formularios
  - 2D / 3D
  - Video / imágenes
  - Tipografía / Documentos
  - Animaciones
  - Speech
- **Silverlight**
  - Subconjunto multiplataforma
  - Multinavegador



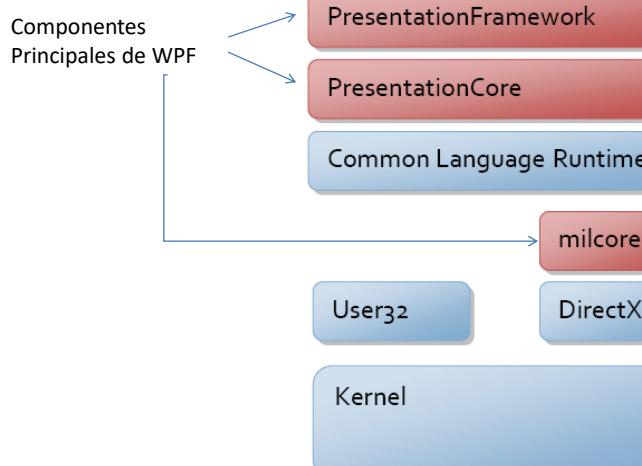
© JMA 2016. All rights reserved.

## Características Avanzadas

- Elementos componibles
- Motor de diseño flexible
- Potente arquitectura de enlace de datos
- Capacidades de impresión, fuentes y documentos
- Controles lookless
- Estilos y plantillas
- La plena integración de todas las disciplinas de la interfaz de usuario

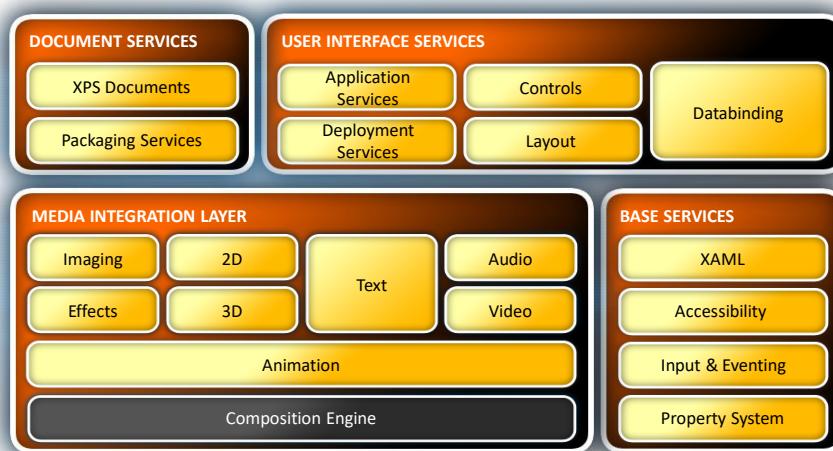
© JMA 2016. All rights reserved.

# Arquitectura WPF



© JMA 2016. All rights reserved.

# Arquitectura WPF

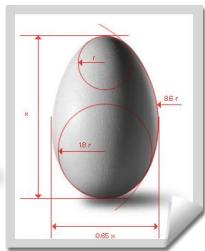


© JMA 2016. All rights reserved.

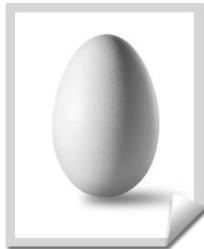
## Workflow Designer x Developers



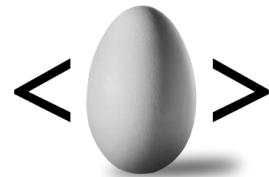
LAYOUT



SPECS



COMP



HTML / CSS

© JMA 2016. All rights reserved

## Workflow Designer x Developers

### PRODUCTION



© JMA 2016. All rights reserved

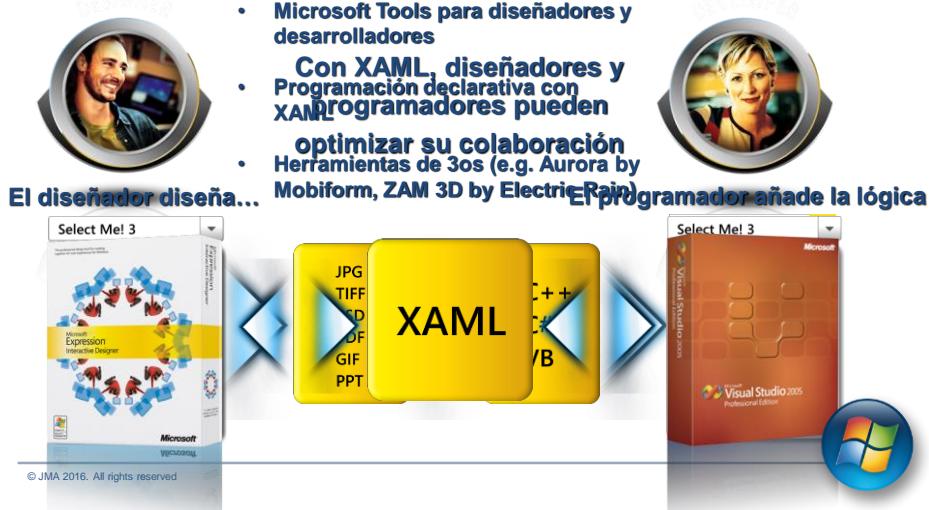
# Workflow Designer x Developers

## FINAL RESULT



© JMA 2016. All rights reserved

## Colaboración diseñador-programador



© JMA 2016. All rights reserved

# ¿Qué es XAML?

## XAML = Extensive Application Markup Language

- Lenguaje declarativo
- Código y diseño separados
- Fácilmente editable desde herramientas



## XAML

- XAML: Extensible Application Markup Language
- Está basado en XML
- Código mas compacto
- Jerárquico
- Soporte para listas
- Así es como se define la UI de las aplicaciones
  - WPF, Silverlight, Workflow Foundation
- Es un mapeo entre XML y los tipos del .NET Framework
  - Nodos -> Tipos
  - Atributos -> CLR Property o Dependency Property

## Sus ventajas

- Se reducen los costos de programación y mantenimiento
- La programación es más eficaz
- Se pueden usar varias herramientas de diseño para implementar y compartir el marcado XAML
- La globalización y localización de las aplicaciones WPF se ha simplificado en gran medida.

© JMA 2016. All rights reserved

## Ejemplo

```
<Canvas  
    xmlns="http://schemas.microsoft.com  
        /client/2007">  
    <TextBlock FontSize="32"  
        Text="Hello world" />  
</Canvas>
```

Hello world

© JMA 2016. All rights reserved

## Markup ⇔ CLR

```
<TextBlock FontSize="32"  
          Text="Hello world" />
```

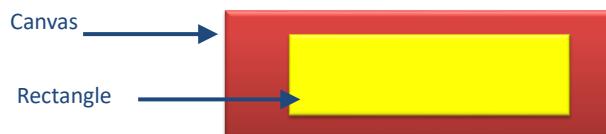
=

```
TextBlock t = new TextBlock();  
t.FontSize = 32;  
t.Text = "Hello world";
```

© JMA 2016. All rights reserved

## Composición

```
<Canvas Width="250" Height="200">  
  <Rectangle  
    Canvas.Top="25" Canvas.Left="25"  
    Width="200" Height="150"  
    Fill="Yellow" />  
</Canvas>
```

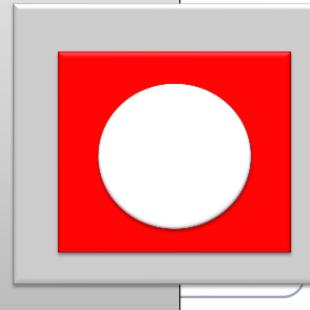


© JMA 2016. All rights reserved

## Composición Relativa

```
<Canvas Background="Light Gray">
    <Canvas Canvas.Top="25" Canvas.Left="25"
        Width="150" Height="100"
        Background="Red">

        <Ellipse Canvas.Top="25"
            Canvas.Left="25"
            Width="150"
            Height="75"
            Fill="White" />
    </Canvas>
</Canvas>
```



## Transformaciones

- Todos los elementos las soportan
- Tipos
  - <RotateTransform />
  - <ScaleTransform />
  - <SkewTransform />
  - <TranslateTransform />
    - Moves
  - <MatrixTransform />
    - Scale, Skew and Translate Combined

## Transformaciones

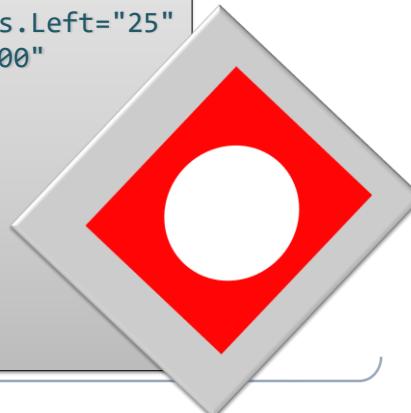
```
<TextBlock Text="Hello World">
    <TextBlock.RenderTransform>
        <RotateTransform Angle="-45" />
    </TextBlock.RenderTransform>
</TextBlock>
```

Hello World

© JMA 2016. All rights reserved.

## Composición y Transformación

```
<Canvas Background="Light Gray">
    <Canvas.RenderTransform>
        <RotateTransform Angle="-45" />
    </Canvas.RenderTransform>
    <Canvas Canvas.Top="25" Canvas.Left="25"
        Width="150" Height="100"
        Background="Red">
        <Ellipse Canvas.Top="25"
            Canvas.Left="25"
            Width="150"
            Height="75"
            Fill="White" />
    </Canvas>
</Canvas>
```



© JMA 2016. All rights reserved.

# XAML

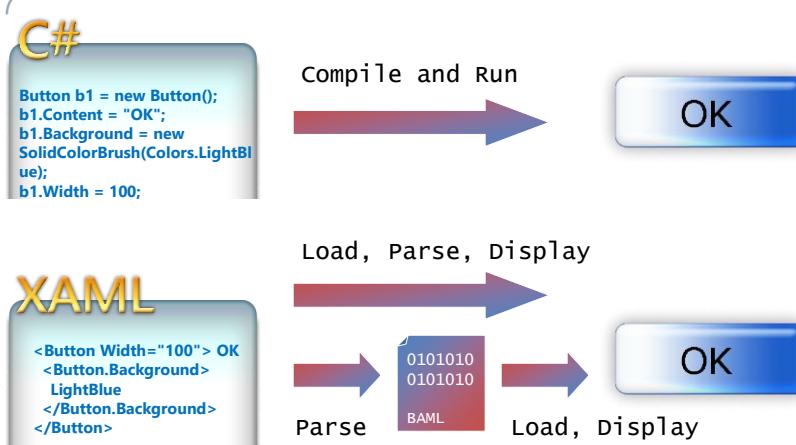
```

10    <!-- XAML basics -->
11
12    <!-- TextBox element maps to System.Windows.Controls.TextBox class
13    Text (Property Attribute) maps to Text property -->
14    <TextBox Text='WPF Demystified' />
15
16    <!-- MouseEnter (Event Attribute) maps to the MouseEnter event.
17        requires a C# or VB code file with MouseEnter procedure. -->
18    <TextBox MouseEnter='TextBox_MouseEnter' />
19
20    <!-- Property Elements are another way to specify property value -->
21    <TextBox>
22        <TextBox.Text>Another example</TextBox.Text>
23    </TextBox>
24
25    <!-- Databinding, Styles, Templates are some of the features
26        enabled with Markup Extensions -->
27
28    <TextBox Text='{Binding AuthorName}' />
29

```

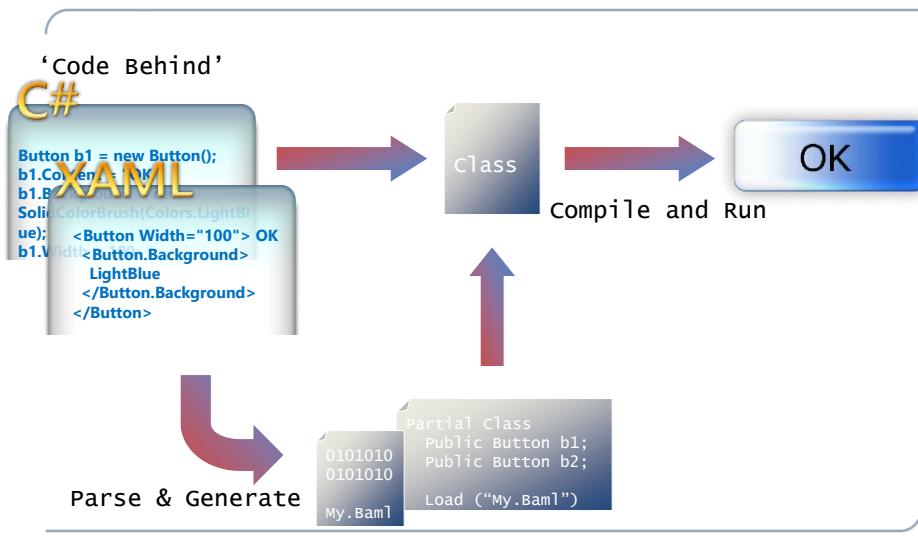
© JMA 2016. All rights reserved.

## ¿XAML o Código?



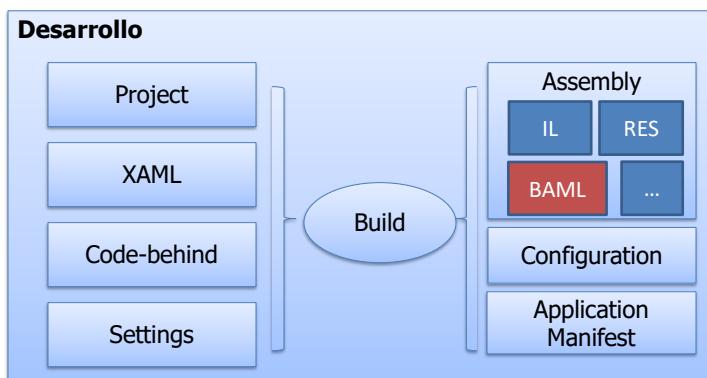
© JMA 2016. All rights reserved.

## ¿XAML o Código?



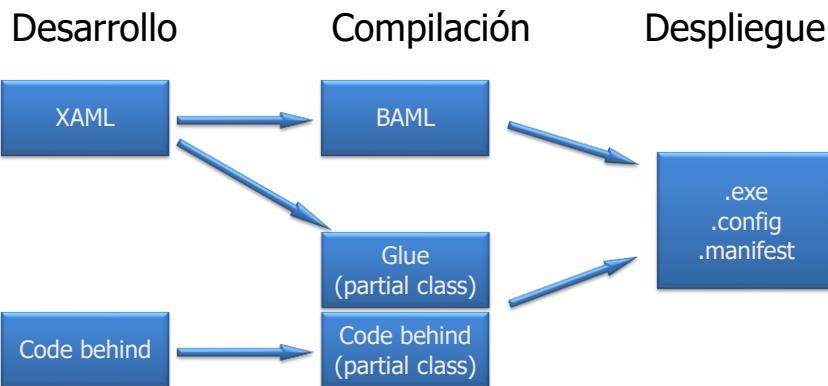
© JMA 2016. All rights reserved

## Anatomía de una aplicación WPF



© JMA 2016. All rights reserved

## Ciclo de vida



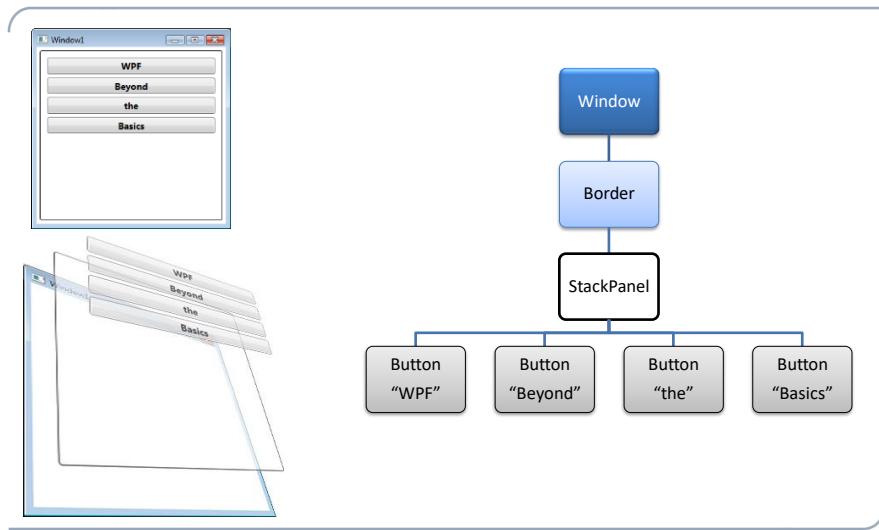
© JMA 2016. All rights reserved

## Ejecución

- Árboles lógicos, visuales y composición
- Rendering basado en capacidades
  - ‘Hardware’ o ‘software’
  - Puede ser detectado usando RenderCapability
- Controla todo el espacio aéreo visual

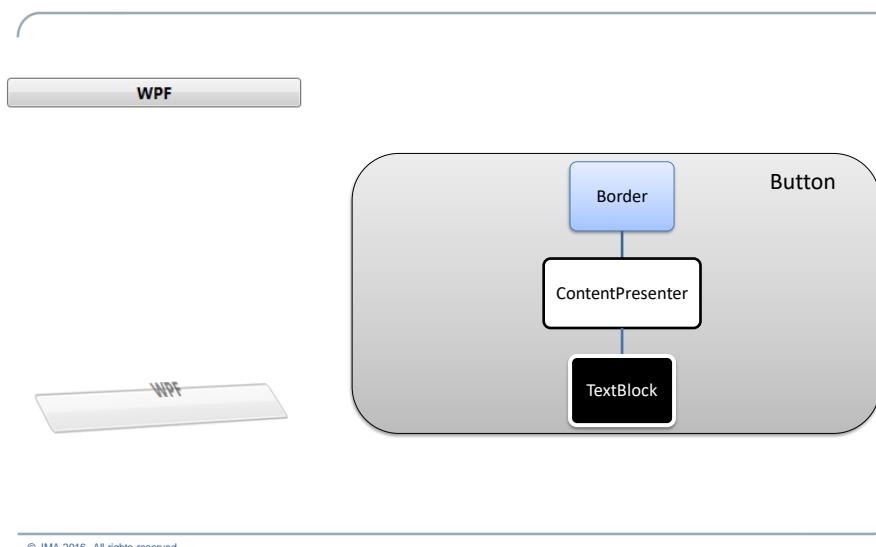
© JMA 2016. All rights reserved

## Árbol Lógico



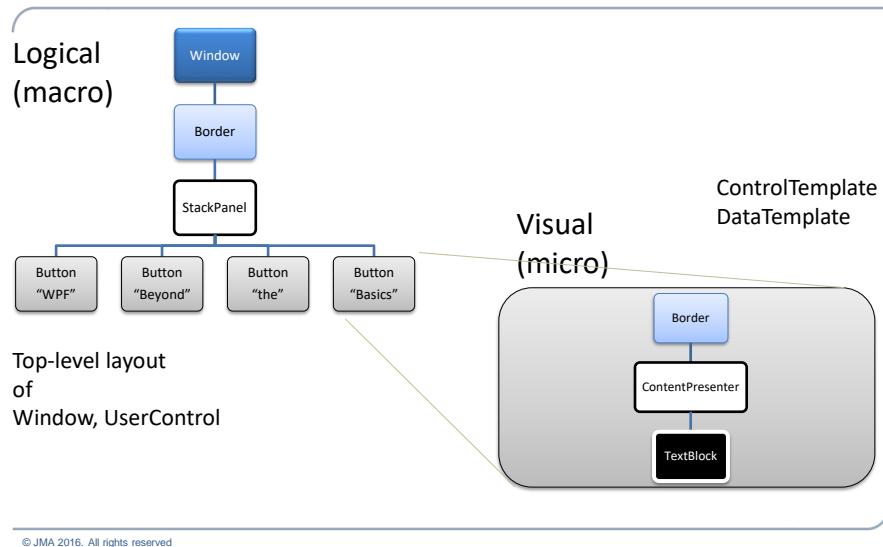
© JMA 2016. All rights reserved

## Árbol Visual

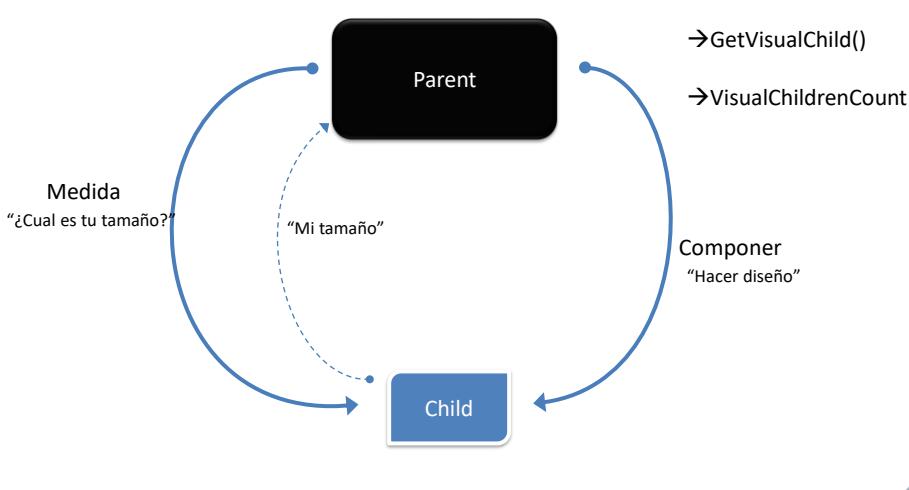


© JMA 2016. All rights reserved

## Logical + Visual

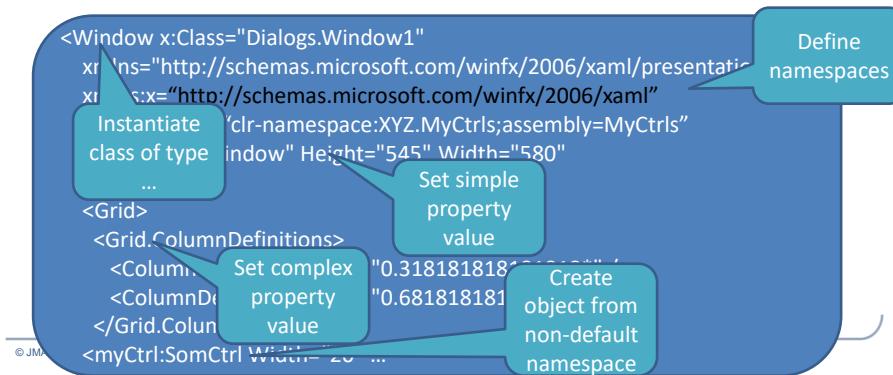


## Layout



# XAML Dinámico

- Se puede leer y guardar XAML desde código
  - XamlReader.Load, XamlReader.Parse
  - XamlWriter.Save



## Sintaxis XAML

- Documentos XML bien formados y validos
- Vocabularios propios
- Vocabulario ampliable a través de los espacios de nombres
- Soporte de extensiones de marcado
- Conversión implícita de cadena al tipo de destino:
  - Valor de propiedad
  - Nombre del controlador de eventos

## Espacios de nombres

- **Básicos:**

```
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:d="http://schemas.microsoft.com/expressionblend/2008"
mc:Ignorable="d"
```

- Para utilizar otros tipos:

```
xmlns:Prefix="clr-namespace:Namespace;assembly=AssemblyName"
```

© JMA 2016. All rights reserved

## Propiedades y eventos

- Son los atributos de la etiqueta
- Propiedades Simples  
`<nombreDeTipo ... propiedad="valor" ... />`
- Propiedades Complejas/Colecciones  
`<nombreDeTipo ... >`  
 `<nombreDeTipo.propiedad>`  
 `...`  
 `<nombreDeTipo.propiedad>`  
 `...`  
`</nombreDeTipo>`
- Propiedades Asociadas  
`<nombreDeTipo ... padre.propiedad="valor" ... />`  
`<nombreDeTipo ... hijos.propiedad="valor" ... />`
- Eventos  
`<nombreDeTipo ... evento="NombreControlador" ... />`

© JMA 2016. All rights reserved

## Extensiones de enlazado

- Las llaves ({ y }) indican el uso de una extensión de marcado que se aparta del tratamiento general de valores de atributo.
  - **Binding**: proporciona un valor enlazado a datos para una propiedad, utilizando el contexto de datos que se aplica al objeto primario en tiempo de ejecución.
  - **RelativeSource**: proporciona información de origen para un objeto Binding que puede navegar por varias posibles relaciones en el árbol de objetos en tiempo de ejecución.
  - **TemplateBinding**: permite que una plantilla de control utilice valores para propiedades con plantilla procedentes de propiedades definidas por el modelo de objetos de la clase que utilizará la plantilla.
  - **StaticResource**: proporciona un valor para una propiedad sustituyendo el valor de un recurso ya definido.
  - **DynamicResource**: proporciona un valor para una propiedad aplazando ese valor para que sea una referencia a un recurso en tiempo de ejecución.
  - **ColorConvertedBitmap**: admite un escenario de creación de imágenes relativamente avanzado.
  - **ComponentResourceKey** y **ThemeDictionary**: admiten aspectos de la búsqueda de recursos y temas que se empaquetan con controles personalizados.

© JMA 2016. All rights reserved

## Extensiones de marcado

- **x:name**
  - Nombre de referencia disponible para todas las etiquetas
  - Utilizado como nombre de las instancias CLR generadas
  - Búsquedas en el árbol: FindName("control")
- **x:key**
  - Identifica de forma exclusiva los elementos que se crean y a los que se hace referencia en un diccionario de recursos.
- **x:class**
  - Enlace al código subyacente
- **x:Type**
  - Construye una referencia Type basada en un nombre de tipo.
- **{x:Static prefix:typeName.staticMemberName}**
  - Referencia a cualquier entidad de código estática por valor definida conforme a Common Language Specification (CLS).
- **{x:Null}**
  - Especifica null como valor para una propiedad.

© JMA 2016. All rights reserved

# Extensiones de Design-Time

- d:DesignHeight and d:DesignWidth
  - d:DesignHeight="300" d:DesignWidth="400"
- d:DataContext
  - d:DataContext="{d:DesignInstance Type=local:Customer}">
- d:DesignInstance and d:IsDesignTimeCreatable
  - <Grid d:DataContext="{d:DesignInstance local:Customer, IsDesignTimeCreatable=True}">
- d:DesignData
  - d:DataContext="{d:DesignData Source=./DesignData/SampleCustomer.xaml}">
- d:DesignSource d:CreateList
  - <CollectionViewSource x:Key="CustomerViewSource">
    - d:DesignSource="{d:DesignInstance local:Customer, CreateList=True}" />
- d>Type

© JMA 2016. All rights reserved

# XML y XAML

- Caracteres especiales
  - Less than (<) &lt;
  - Greater than (>) &gt;
  - Ampersand (&) &amp;
  - Quotation mark ("") &quot;
- No preserva el espacio en blanco (sp, br, tab, ...)
- Para preservar el espacio en blanco:  
`<nombreDeTipo xml:space="preserve" ... >`  
`...`  
`</nombreDeTipo>`

© JMA 2016. All rights reserved

## Diseño de aplicaciones y formularios

- Estructura de una aplicación
- Objeto aplicación
- Layout o contenedores
- Controles
- Decoración y Cosmética de aplicaciones
- Animaciones

© JMA 2016. All rights reserved

## Despliegue WPF

- Ensamblado .NET
  - Ejecutable tradicional Setup, ClickOnce
- Aplicación XBAP
  - Dentro del navegador
  - Modelo de navegación integrado con browser
- Loose XAML
  - Renderización directa en browser
  - Opciones interesantes: ASP.NET / XML + XSL
- Documento
  - Formato de documento XPS = Subset XAML

© JMA 2016. All rights reserved

# Modelos

- SDI: interfaz de documento único
  - Ventana principal (Window):
    - Controles (UserControl)
    - Ventanas emergentes (PopUP)
- MDI: interfaz de múltiples documentos
  - Soportado por herramientas de terceros.
- Navegación: tipo web
  - Ventana de navegación (NavigationWindow).
    - Páginas de contenido (Page) e hipervínculos (Hyperlink)
    - Marcos (Frame) y motor de navegación (NavigationService)

© JMA 2016. All rights reserved

## Modelo de Navegación

- Page: Encapsula una página de contenido a la que pueda navegar, se diseña de forma similar a las ventanas.
- Hyperlink: Permite que un usuario inicie la navegación en un objeto Page determinado.
- NavigationService: Encargado de la búsqueda y descarga de la página.
- Diario (Journal): Implementa un servicio del historial de navegación que almacena una entrada para cada fragmento de contenido al que se ha navegado previamente.
- NavigationWindow: Proporciona una ventana principal para aplicaciones independientes de navegación.
- Frame: Proporciona un contenedor de páginas para su uso en ventanas u otras páginas.

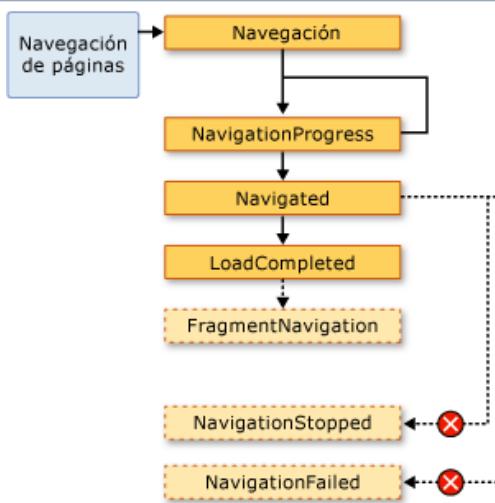
© JMA 2016. All rights reserved

# Navegación

- Navegación por hipervínculos  
`<Hyperlink NavigateUri="pag2.xaml">...</Hyperlink>`
- Navegación por fragmentos  
`<TextBlock Name="Fragmento1">...</TextBlock>`  
`<Hyperlink  
NavigateUri="pag2.xaml#Fragmento1">...</Hyperlink>`
- Navegación por código :  
`ns = NavigationService.GetNavigationService(refPage);`  
`ns = this.NavigationService;`  
`ns.Navigate(new Page2());`  
`ns.Navigate(new Uri("Page2.xaml", UriKind.Relative));`

© JMA 2016. All rights reserved

## Duración de la navegación



© JMA 2016. All rights reserved

## Aplicación (objeto Application)

- Crear y administrar la infraestructura común de las aplicaciones.
- Realizar el seguimiento e interactuar con la duración de la aplicación.
- Recuperar y procesar los parámetros de la línea de comandos.
- Compartir propiedades y recursos del ámbito de la aplicación.
- Detectar y responder a las excepciones no controladas.
- Devolver códigos de salida.
- Administrar las ventanas en las aplicaciones.
- Realizar el seguimiento y administrar la navegación.
- Se define en:

App.xaml

App.cs

```
public partial class App : Application { }
```

© JMA 2016. All rights reserved

## Objeto inicial

- Pagina inicial:  
`<Application ... StartupUri="MainPage.xaml" ... />`
- Ventana principal:  
`<Application ... StartupUri="MainWindow.xaml" ... />`  
`Application.MainWindow`
- Evento:  
`<Application ... Startup="app_Startup" ... />`
- SplashScreen
  - Imagen WIC (BMP, GIF, JPEG, PNG o TIFF)
    - Propiedades→Acción de compilación=SplashScreen

© JMA 2016. All rights reserved

## Cierre de la aplicación

- Para facilitar la administración del cierre de la aplicación, Application proporciona el método Shutdown, la propiedad ShutdownMode y los eventos SessionEnding y Exit.
- Modo de apagado (ShutdownMode):
  - OnLastWindowClose
  - OnMainWindowClose
  - OnExplicitShutdown

© JMA 2016. All rights reserved

## Diseño (Layout)

- **Canvas:**
  - los controles secundarios proporcionan su propio diseño.
- **DockPanel:**
  - los controles secundarios se alinean con los bordes del panel.
- **Grid:**
  - los controles secundarios se sitúan por filas y columnas.
- **StackPanel:**
  - los controles secundarios se apilan vertical u horizontalmente.
- **VirtualizingStackPanel:**
  - los controles secundarios se organizan en una vista "virtual" de una sola línea en sentido horizontal o vertical.
- **WrapPanel:**
  - los controles secundarios se sitúan por orden de izquierda a derecha y se ajustan a la línea siguiente cuando hay más controles de los que caben en la línea actual.

© JMA 2016. All rights reserved

# Controles de WPF por función

- **Diseño:**
  - Border, BulletDecorator, Canvas, DockPanel, Expander, Grid, GridView, GridSplitter, GroupBox, Panel, ResizeGrip, Separator, ScrollBar, ScrollViewer, StackPanel, Thumb, Viewbox, VirtualizingStackPanel, Window y WrapPanel.
- **Navegación:**
  - Frame, Hyperlink, Page, NavigationWindow y TabControl.
- **Botones:**
  - Button y RepeatButton.
- **Menús:**
  - ContextMenu, Menu y ToolBar.
- **Entrada:**
  - TextBox, RichTextBox y PasswordBox.
- **Selección:**
  - CheckBox, ComboBox, ListBox, RadioButton y Slider.

© JMA 2016. All rights reserved

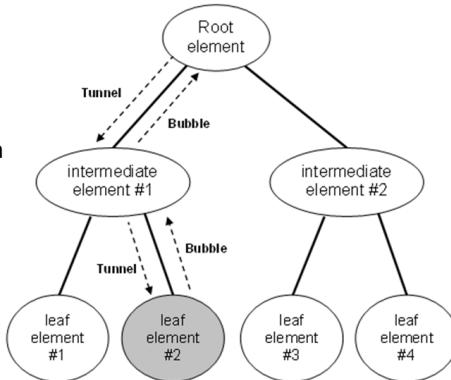
# Controles de WPF por función

- **Presentación y selección de fechas:**
  - Calendar y DatePicker.
- **Presentación de datos:**
  - DataGrid, ListView y TreeView.
- **Cuadros de diálogo:**
  - OpenFileDialog, PrintDialog y SaveFileDialog.
- **Información para el usuario:**
  - AccessText, Label, Popup, ProgressBar, StatusBar, TextBlock y ToolTip.
- **Documentos:**
  - DocumentViewer, FlowDocumentPageViewer, FlowDocumentReader, FlowDocumentScrollView y StickyNoteControl.
- **Entradas de lápiz digitales:**
  - InkCanvas y InkPresenter.
- **Multimedia:**
  - Image, MediaElement y SoundPlayerAction.

© JMA 2016. All rights reserved

## Eventos en WPF

- Eventos enrutados:
  - Propagación: se invocan los controladores de eventos en el origen del evento.
  - Directo: sólo el propio elemento de origen tiene la oportunidad de invocar controladores como respuesta
  - Túnel: inicialmente, se invocan los controladores de eventos en la raíz de árbol de elementos.
- Manejar eventos desde código .NET
- Triggers



© JMA 2016. All rights reserved

## Comandos en WPF

- Propiedades: Command y CommandParameter
- Implementación de ICommand:
  - Execute, CanExecute, CanExecuteChanged
- DelegateCommand  
<http://www.wptutorial.net/delegatecommand.html>
- Comandos enrutados
- Built-in commands
  - ApplicationCommands – Close, Copy, Cut, ...
  - ComponentCommands – MoveDown, PageUp, ...
  - MediaCommands – Play, Stop, ...
  - NavigationCommands – Back, Forward, ...

© JMA 2016. All rights reserved

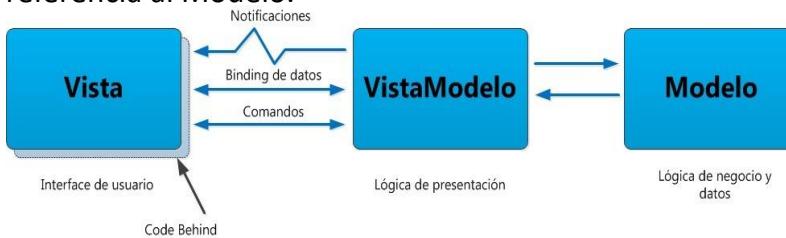
## Enlace a datos

- Extensiones de marcado como formas de enlace a datos
- Patrón MVVM (Model-View-ViewModel)
- ViewModel: IObservable, ICommand
- Enlace a datos (Binding)
- Objetos de negocio y documentos XML
- Colecciones de datos
- Conversión y validación de datos

© JMA 2016. All rights reserved

## Patrón de Diseño Model View ViewModel (MVVM)

- El **Modelo** es la entidad que representa el concepto de negocio.
- La **Vista** es la representación gráfica del control o un conjunto de controles que muestran el Modelo de datos en pantalla.
- La **VistaViewModel** es la que une todo. Contiene la lógica del interfaz de usuario, los comandos, los eventos y una referencia al Modelo.



© JMA 2016. All rights reserved

## Características MVVM

- La vista y la VistaModelo se comunican mediante enlaces de datos, métodos, propiedades, eventos y mensajes.
- La VistaModelo expone propiedades y comandos además de modelos.
- La vista se encarga de sus propios eventos relacionados con la interface al usuario y los pasa al modelo de vista mediante comandos.
- Los modelos y las propiedades de la VistaModelo son actualizados desde la vista usando enlaces de datos bidireccionales.

© JMA 2016. All rights reserved

## ¿Cuáles son los beneficios del patrón MVVM?

- Separación de vista / presentación.
- Permite las pruebas unitarias: como la lógica de presentación está separada de la vista, podemos realizar pruebas unitarias sobre la VistaModelo.
- Mejora la reutilización de código.
- Soporte para manejar datos en tiempo de diseño.
- Múltiples vistas: la VistaModelo puede ser presentada en múltiples vistas, dependiendo del rol del usuario por ejemplo.

© JMA 2016. All rights reserved

## Patrones e Interfaces

- Observable
  - IObservable
  - IObservableCollection
  - INotifyPropertyChanged
- Command
  - ICommand
- Validación
  - IDataErrorInfo

© JMA 2016. All rights reserved

## Frameworks de MVVM

- Prism – Microsoft
- MVVM Light
- Simple MVVM
- Caliburn
- [Kit de herramientas de MVVM de la comunidad .NET](#)
- [ReactiveUI](#)
- [Prism Library](#)

© JMA 2016. All rights reserved

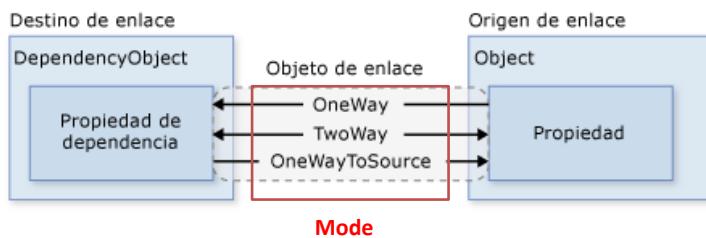
## Enlace de datos (DataBinding)

- El enlace de datos es el proceso que establece una conexión entre la UI de la aplicación y la lógica del negocio.
- Se implementa mediante objetos que actúan de pasarela interceptando los eventos.
- Ventajas del DataBinding:
  - Un mayor número de propiedades que admiten de forma inherente el enlace de datos.
  - Una representación flexible de los datos en la UI.
  - La separación bien definida de la lógica del negocio de la UI.

© JMA 2016. All rights reserved

## Dirección del flujo de datos

- Se controla este comportamiento estableciendo la propiedad Mode del objeto Binding



© JMA 2016. All rights reserved

## Conceptos avanzados

- Behaviors
- Conversores
- Estilos y Temas
- Plantillas visuales (Control Templates)
- Plantillas de datos (Data Templates)
- Nuevos controles
- Animaciones, transformaciones
- Documentos: FlowDocument

© JMA 2016. All rights reserved

<http://msdn.microsoft.com/es-es/library/wtzawcsz%28v=vs.100%29.aspx>

## IMPLEMENTAR APLICACIONES Y COMPONENTES

© JMA 2016. All rights reserved

# Estrategias de implementación

- Windows Installer
  - Permite crear paquetes del instalador que se distribuyan a los usuarios.
  - Mayor flexibilidad en los tipos de instalaciones y control en el proceso de instalación.
- ClickOnce
  - Permite implementar aplicaciones de consola y de actualización automática en Windows, que pueden instalarse, actualizarse y ejecutarse desde un sitio web.
  - Es la mejor opción para aquellas que requieren actualizaciones frecuentes.
  - Pero los usuarios deben tener conectividad de red para aprovechar las funciones de actualización y pueden presentar problemas con los niveles de confianza.
- InstallShield
  - Permite crear instalaciones mediante InstallShield 2010 Limited Edition.

© JMA 2016. All rights reserved

## Windows Installer - Tipos

- Instalación Windows:
  - Instalador (.msi) para una aplicación basada en Windows donde los archivos se instalan en el directorio Archivos de programa de los equipos de los usuarios finales.
- Instalación Web:
  - Instalador (.msi) para una aplicación Web donde los archivos se instalan en un directorio Raíz virtual de un servidor web.
- Módulo de combinación:
  - Archivo .msm que empaqueta los componentes, que pueden utilizar varias aplicaciones basadas en Windows, en un mismo módulo, que pueden incluirse en cualquier proyecto de implementación lo que facilita compartirlos.
- Archivo CA:
  - Archivo CAB que empaqueta componentes en un único fichero comprimido que puede descargarse de un servidor web a un explorador web.

© JMA 2016. All rights reserved

## Windows Installer - Elementos

- **Sistema de archivos:** agregar resultados del proyecto, archivos y otros elementos a la implementación, y especificar el lugar del equipo de destino donde se van a instalar estos elementos.
- **Registro:** especificar los valores y claves que se van a agregar al Registro del equipo de destino.
- **Tipos de archivos:** establecer asociaciones de extensiones de archivo en el equipo de destino.
- **Interfaz de usuario:** especificar y establecer las propiedades de los cuadros de diálogo predefinidos que se muestran durante la instalación en el equipo de destino.
- **Acciones personalizadas:** especificar acciones adicionales que se van a realizar en el equipo de destino al final de la instalación.
- **Condiciones de inicio:** especificar las condiciones que deben cumplirse para que la instalación se ejecute correctamente.

© JMA 2016. All rights reserved

## ClickOnce

- Una aplicación ClickOnce es cualquier aplicación de Windows Presentation Foundation (.xbap), de Windows Forms (.exe) o de consola (.exe) o una solución de Office (.dll) publicada mediante la tecnología ClickOnce.
- Se puede publicar de tres maneras distintas: desde una página web, desde un recurso compartido de archivos de red o desde otros medios como un CD-ROM.
- Una aplicación ClickOnce se puede instalar en el equipo de un usuario final y ejecutarse localmente incluso cuando el equipo está trabajando sin conexión.
- Las aplicaciones ClickOnce se pueden actualizar automáticamente; pueden comprobar si hay versiones más recientes cuando se publican y reemplazar automáticamente los archivos actualizados.

© JMA 2016. All rights reserved

## Publicar con ClickOnce

- Pasos previos opcionales:
  - Establecer “Información del ensamblado”
  - Firmar y establecer los permisos (Seguridad) del ensamblado.
- Publicar:
  - Especificar la ubicación y la dirección URL de la carpeta de publicación.
  - Establecer el modo: “La aplicación sólo está disponible en línea” o “La aplicación también está disponible sin conexión”.
  - Configurar la instalación.
  - Lanzar la publicación: Publicar ahora.

© JMA 2016. All rights reserved

## Configurar ClickOnce

- **Archivos de aplicación:** especifica cómo y dónde se instalan los archivos individuales.
- **Requisitos previos:** especifica los componentes necesarios que se instalarán junto con la aplicación y desde donde.
- **Actualizaciones de la aplicación:** especifica si la aplicación debe buscar actualizaciones (antes o después de que se inicie la aplicación), la frecuencia con la que buscará actualizaciones y la versión mínima requerida.
- **Opciones de publicación:** especifica opciones adicionales de publicación avanzada.

© JMA 2016. All rights reserved

## dotnet cli

- La implementación dependiente de la plataforma genera un archivo .dll multiplataforma que usa el entorno de ejecución de .NET instalado localmente.
  - dotnet publish -c Release -p:UseAppHost=false
- La implementación dependiente de la plataforma genera un archivo ejecutable específico de la plataforma que usa el entorno de ejecución de .NET instalado localmente.
  - dotnet publish -c Release -r <RID> --self-contained false
- La implementación autocontenido genera un archivo ejecutable específico de la plataforma e incluye una copia local del entorno de ejecución de .NET.
  - dotnet publish -c Release -r <RID> --self-contained true
- RID es la abreviatura en inglés de identificador de runtime y se usan para identificar las plataformas de destino donde se ejecuta la aplicación:
  - win-x64, win-x86, win-arm64,
  - linux-x64, linux-musl-x64, linux-musl-arm64, linux-arm, linux-arm64, linux-bionic-arm64, osx-x64, osx-arm64,
  - ios-arm64, android-arm64

© JMA 2016. All rights reserved

## Dockerfile

#See <https://aka.ms/customizecontainer> to learn how to customize your debug container and how Visual Studio uses this Dockerfile to build your images for faster debugging.

```
FROM mcr.microsoft.com/dotnet/aspnet:6.0 AS base
WORKDIR /app
EXPOSE 80
EXPOSE 443

FROM mcr.microsoft.com/dotnet/sdk:6.0 AS build
WORKDIR /src
COPY ["Contenedores/Contenedores.csproj", "Contenedores/"]
RUN dotnet restore "Contenedores/Contenedores.csproj"
COPY .
WORKDIR "/src/Contenedores"
RUN dotnet build "Contenedores.csproj" -c Release -o /app/build

FROM build AS publish
RUN dotnet publish "Contenedores.csproj" -c Release -o /app/publish /p:UseAppHost=false

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "Contenedores.dll"]
```

© JMA 2016. All rights reserved