



© 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 Framework?

- 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.
- .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

Implementaciones de .NET

- 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 cuatro implementaciones principales de .NET que Microsoft desarrolla y mantiene activamente:
 - .NET Framework
 - .NET Core
 - Mono
 - UWP

© 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.
- 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

- 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

Mono

- Mono es una implementación de .NET 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

.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

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



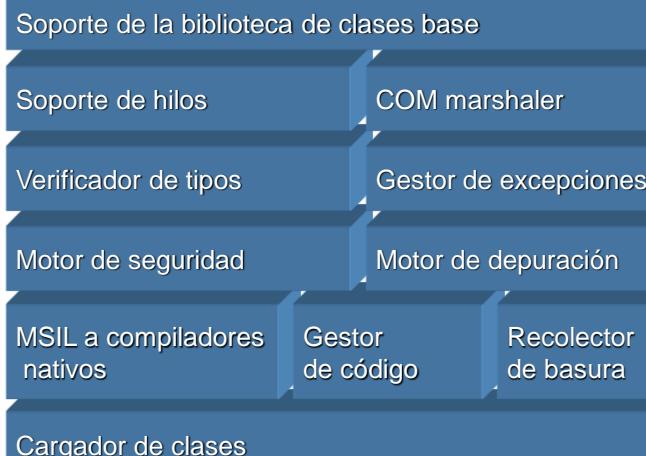
© JMA 2016. All rights reserved

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)

© JMA 2016. All rights reserved

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

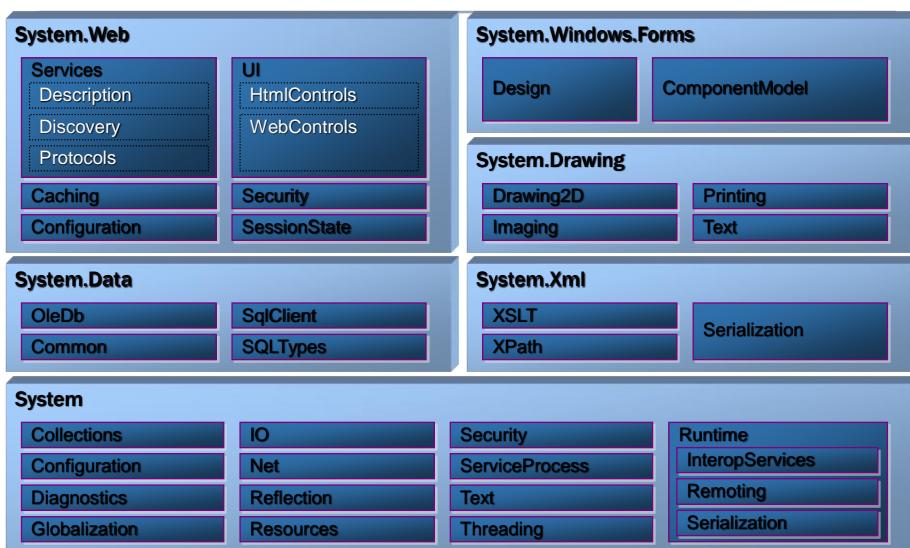
© JMA 2016. All rights reserved

.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
 - ...

© JMA 2016. All rights reserved

.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

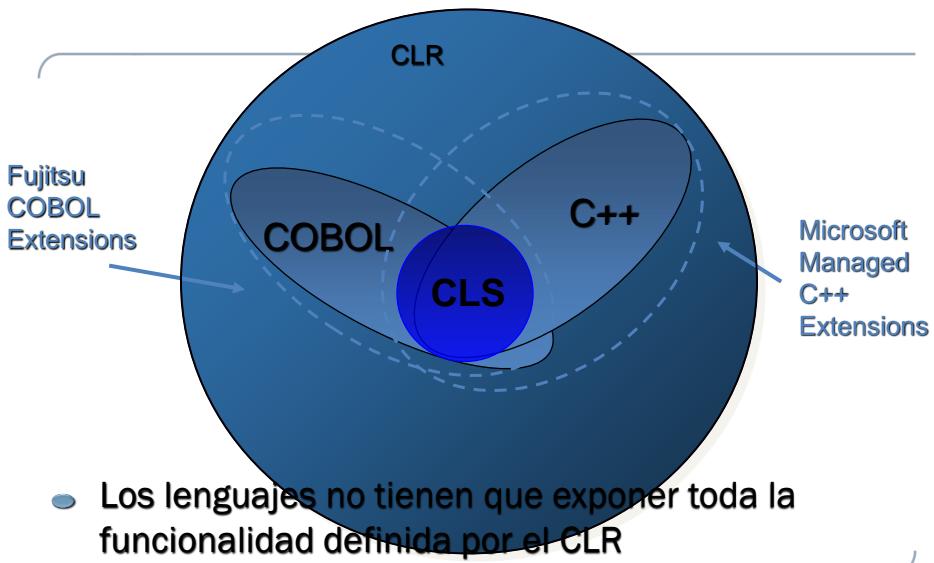
Lenguajes .NET

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

Lenguajes .NET



© 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

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

VB.NET

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

C#

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

C++

© 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.
INVOKE 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 xsqlcmd
Cmd x='authors'
cmd = sqlcmd("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

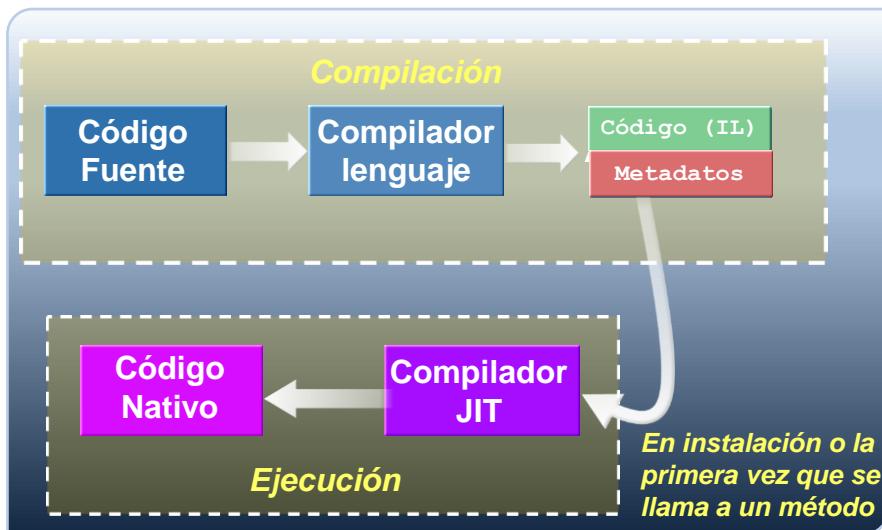
© JMA 2016. All rights reserved

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

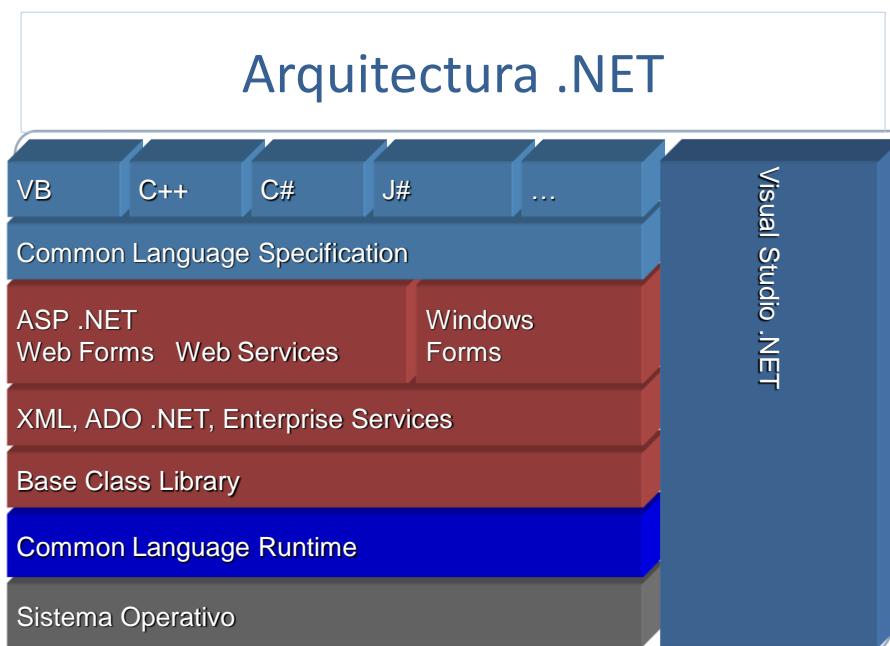
- Tipos (según su visibilidad):
 - Privados: Se almacenan en el directorio de la aplicación o en uno de sus subdirectorios.
 - Compartidos: Se almacenan en el GAC (Caché de ensamblados global). Control de versiones, seguros (firmados con nombre seguro), eficientes (ya verificados y rápida localización).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



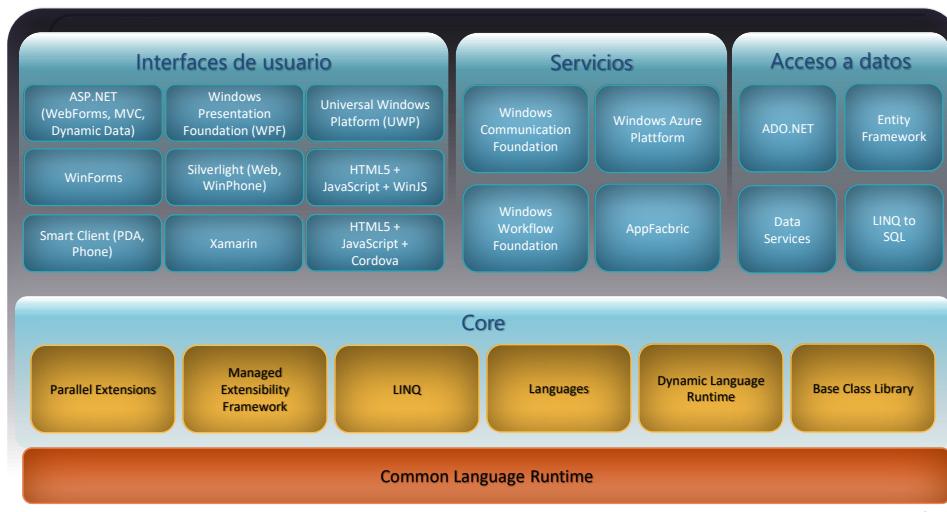
© JMA 2016. All rights reserved

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?

	Cliente	Servidor
Aplicación de Escritorio	✓	✓ *
Aplicación Web		✓
Aplicación de Consola	✓	✓ *
Aplicación Móvil	.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

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

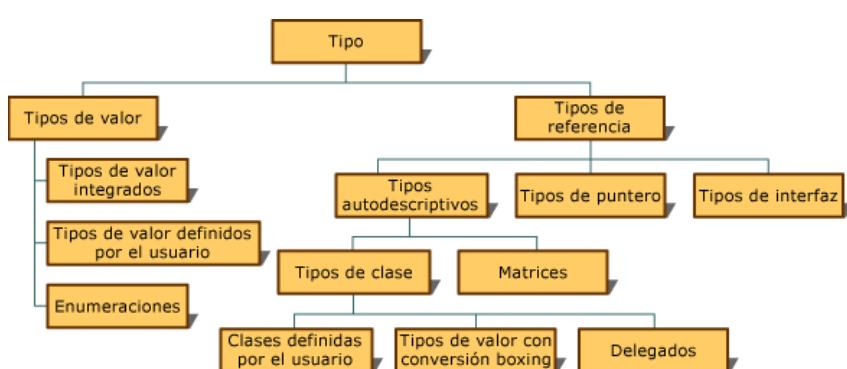
© JMA 2016. All rights reserved

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**): \$"...{expresión}..."
 - 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>
#define <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 numéricicas

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

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

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

© JMA 2016. All rights reserved

Expresiones y Operadores

Operador	Descripción
<code><expr1> + <expr2></code>	Suma operandos aritméticos. Concatenación de cadenas. Combinación de delegados.
<code><expr1> - <expr2></code>	Resta operandos aritméticos. Elimina la asociación de delegados.
<code><expr1> * <expr2></code>	Multiplica operandos aritméticos.
<code><expr1> / <expr2></code>	Divide operandos aritméticos. El resultado será real o entero en función de los operandos.
<code><expr1> % <expr2></code>	Modulo (resto) de la división entera.
<code>++<variable></code>	Preincremento: incrementa en 1 el valor de la variable antes de consultar su valor.
<code><variable>++</code>	Postincremento: incrementa en 1 el valor de la variable después de consultar su valor.
<code>- <variable></code>	Predecremento: decrementa en 1 el valor de la variable antes de consultar su valor.
<code><variable>- -</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><expr1> & <expr2></code>	AND binario para operadores de tipos enteros y lógico para operadores booleanos.
<code><expr1> <expr2></code>	OR binario para operadores de tipos enteros y lógico para operadores booleanos.
<code><expr1> ^ <expr2></code>	XOR binario para operadores de tipos enteros y lógico para operadores booleanos.
<code><expr> << <contador></code>	Desplaza a la izquierda sin desbordamiento los bit de la expresión rellenado con ceros las posiciones libres.
<code><expr> >> <contador></code>	Desplaza a la derecha sin desbordamiento los bit de la expresión rellenado con ceros las posiciones libres.
<code>~<expresión></code>	Complemento binario. Negación bit a bit.
<code>!<expresión></code>	Negación lógica.
<code><expr1> && <expr2></code>	AND lógico cortocircuitado, solo evalúa la segunda expresión si la primera es true.
<code><expr1> <expr2></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><expr1> <comp> <expr2></code>	Compara dos expresiones, donde <code><comp></code> puede ser: <code>==</code> (igual), <code>!=</code> (distinto), <code><</code> (menor), <code><=</code> (menor o igual), <code>></code> (mayor), <code>>=</code> (mayor o igual).
<code><expresión> is <tipo></code>	Comprueba si el tipo en tiempo de ejecución de un objeto es compatible con un tipo dado
<code><cond>?<exp1>:<exp2></code>	Condicional: devuelve el primer valor sin la condición es true o el segundo si es false.
<code><objeto>?.<miembro></code>	Condicional de NULL (ver. 6.0): si el objeto es nulo devuelve nulo, sino el resultado de invocar la miembro, equivale a <code><objeto> == null ? null : <objeto>.<miembro></code>
<code><expr1>??<expr2></code>	Condicional: devuelve el primer valor si no es nulo o el segundo si el primero es nulo. (Valor de sustitución del null)
<code><expresión> as <tipo></code>	Devuelve conversión de referencias entre tipos compatibles o null si no es compatible.
<code>(<tipo>) <expresión></code>	Realiza conversiones explícitas entre tipos compatibles.

© JMA 2016. All rights reserved

Expresiones y Operadores

Operador	Descripción
<code><dest> = <expr></code>	Asigna la expresión al destinatario.
<code><dest> <opr>= <expr></code>	Equivale a <code><destinatario> = <destinatario> <opr> <expr></code> Donde el <code><opr></code> puede ser: <code>*</code> / <code>%</code> <code>+</code> <code>-</code> <code><<</code> <code>>></code> <code>&</code> <code>^</code> <code> </code>
<code>(...) => <expresión></code>	Función anónima (expresión lambda) (ver. 4.0)
<code>delegate {...}</code>	Función anónima (método anónimo)
<code>nameof(<miembro>)</code>	(ver. 6.0) 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>&<expresión></code>	Dirección de memoria de su operando. [No administrado].
<code><expr1>-><expr2></code>	Acceso a un miembro de la estructura apuntada. El primer operador debe ser un puntero. Similar a <code>(*<expr1>).<expr2></code> . [No administrado].
<code>sizeof(<tipo>)</code>	Obtiene el tamaño en bytes de un tipo de valor.
<code>stackalloc <tipo>[<tamaño>]</code>	Asigna un bloque de memoria en la pila y devuelve un puntero. [No administrado].

© 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

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> <nombreColección>;
<tipoColeccion> <nombreColección>= new <tipoColección>();
<tipoColección> <nombreColección>= new <tipoColección>(<tamañoInicial>)
<tipoColección> <nombreColección>= new <tipoColección>(<otraColección>);
<tipoColección> <nombreColección>= new <tipoColección> {<valores>}; (ver. 3.0)
```

© 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

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
new	Ocultación o sombreado	Ocultar o sombra un elemento heredado de una clase base.
abstract	Abstracta	Marca la clase como abstracta, no es instanciable. Incompatible con sealed
sealed	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

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 [DllImport("???dll")].

© 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 subprocesso 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)
 - 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:


```
[attributes] [modifiers] <tipoRetorno> [<interfaz>.]<nombreFunción>(<ListaDeParámetros>) { ... }
[attributes] [modifiers] void [<interfaz>.]<nombreProcedimiento>(<ListaDeParámetros>) { ... }
[attributes] [modifiers] <tipoRetorno> [<interfaz>.]<nombreFunción>(<ListaDeParámetros>);
[attributes] [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:


```
[attributes] [new] <tipoRetorno> [<interfaz>.]<nombreFunción>(<ListaDeParámetros>);
[attributes] [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.

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

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

-
- [atributes] [modifiers] type <interfaz>.this [formal-index-parameter-list]
 {<DescriptoresDeAcceso>}
- [atributes] [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:


```
[atributes] [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.

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.

`<TipoDelegado> <Variable> = new
 <TipoDelegado>(<Objeto>.<Método>); // Antes del 2.0
 <TipoDelegado> <Variable> = <Objeto>.<Método>;`
- Donde la firma de `<Objeto>.<Método>` debe coincidir con la declarada en `<TipoDelegado>`.
- Métodos anónimos:

`delegate([formal-parameters]) { ... Cuerpo... }`

© 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

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 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.
- Se puede realizar la inicialización de objetos sin llamadas explícitas a un constructor: (ver. 3.0)
`Clase C = new Clase { Propiedad1=v1, P2=v2 };`
- Si se marcan como privados impiden la instanciación desde fuera de la clase.

- 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

ELEMENTOS AVANZADOS

© 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

Componentes 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
`[attributes] [modifiers] class nombre<listaAliasDeTipo> [:base-list] [lista de restricciones de tipo]{... Miembros ...}[:]`
- Declaraciones de estructuras genéricas
`[attributes] [modifiers] struct nombre<listaAliasDeTipo> [:ListaDeInterfces] [lista de restricciones de tipo]{... Miembros ...}[:]`
- Declaraciones de Interfaces genéricas
`[attributes] [modifiers] interface nombre<listaAliasDeTipo> [:ListaDeInterfces] [lista de restricciones de tipo]{... Miembros ...}[:]`

© JMA 2016. All rights reserved

Componentes 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

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

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.
 - 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

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

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

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

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.


```
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 rs1t = unnamed.Item2; // two
var rs1t = named.name; // one
```
- Las tuplas pueden ser tipos de retorno o de parámetros:


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

© 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 de tupla 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..

© 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

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.
 - 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

Tipos de registro (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

Tipos de registro (ver.9.0)

- La declaración:

```
public record Person(string FirstName, string LastName);  
public 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

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.
- 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 y son colecciones con seguridad de tipos en tiempo de compilación. Debido a esto, las colecciones genéricas 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 subprocessos con el fin de obtener acceso a los elementos de la colección desde varios subprocessos. Las clases de colección inmutables en el espacio de nombres System.Collections.Immutable (NuGet package) son intrínsecamente seguras para los subprocessos, 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

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

© JMA 2016. All rights reserved

Selección de una colección

- Para almacenar elementos como pares clave/valor para una consulta rápida por clave:
 - Hashtable, Dictionary<TKey, TValue>, ConcurrentDictionary<TKey, TValue>,
ReadOnlyDictionary<TKey, TValue>, ImmutableList<TValue>
- Con acceso a elementos por índice:
 - Array, ArrayList, List<T>, ImmutableList<T>, ImmutableArray
- Con estructura de cola FIFO (el primero en entrar es el primero en salir):
 - Queue, Queue<T>, ConcurrentQueue<T>, ImmutableList<T>
- Con estructura de pila LIFO (el último en entrar es el primero en salir):
 - Stack, Stack<T>, ConcurrentStack<T>, ImmutableList<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>, ImmutableList<TValue>, ImmutableList<T>
- Un conjunto (sin repetición):
 - HashSet<T>, SortedSet<T>, ImmutableList<TValue>, ImmutableList<T>

© 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

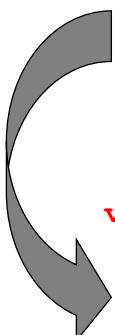
Lista de providers

- 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()

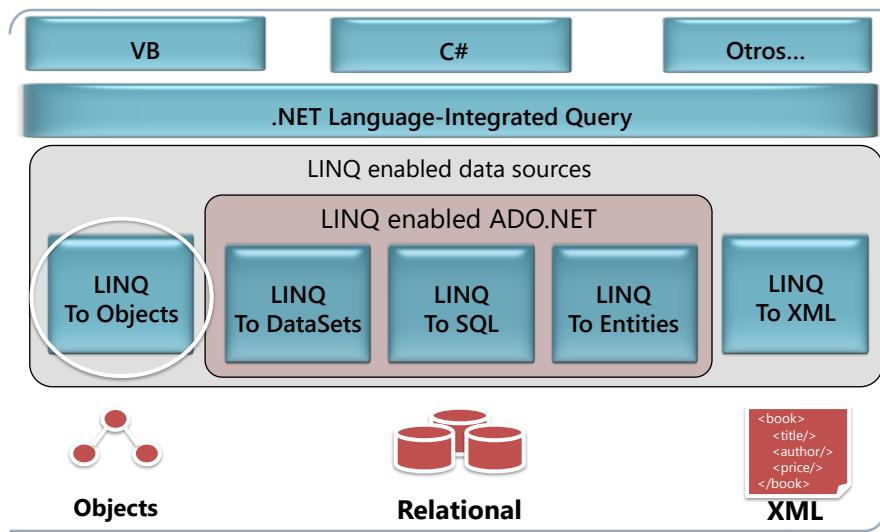
© JMA 2016. All rights reserved

Expresiones de consulta

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

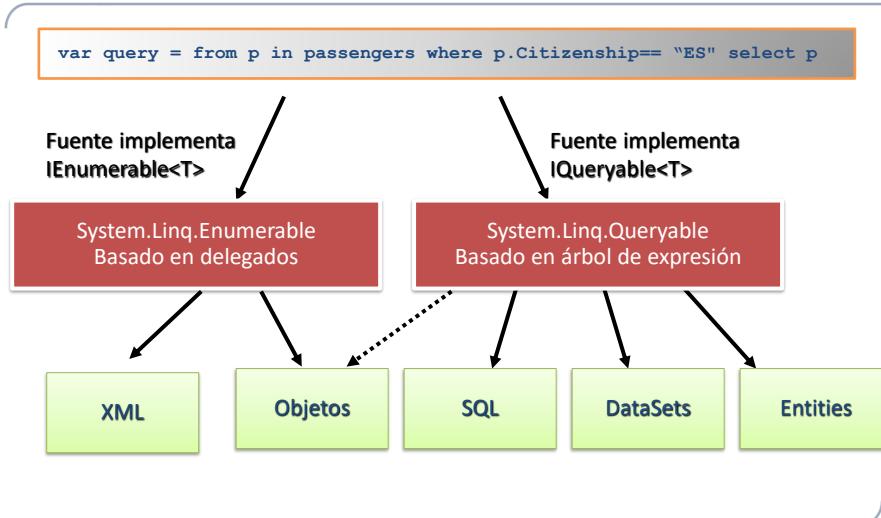
© JMA 2016. All rights reserved

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<T></code>	Basados en <code>IQueryable<T></code>
Interfaz	<code>IEnumerable<T></code>	<code>IQueryable<T></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>
Mis 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.

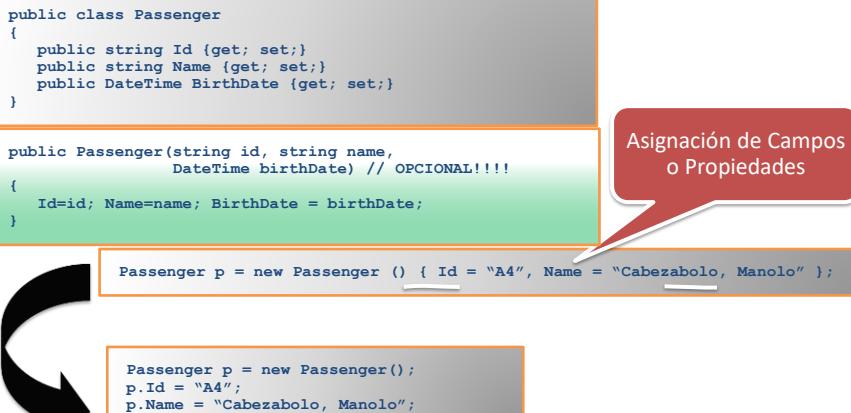
© JMA 2016. All rights reserved

Nuevas características

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

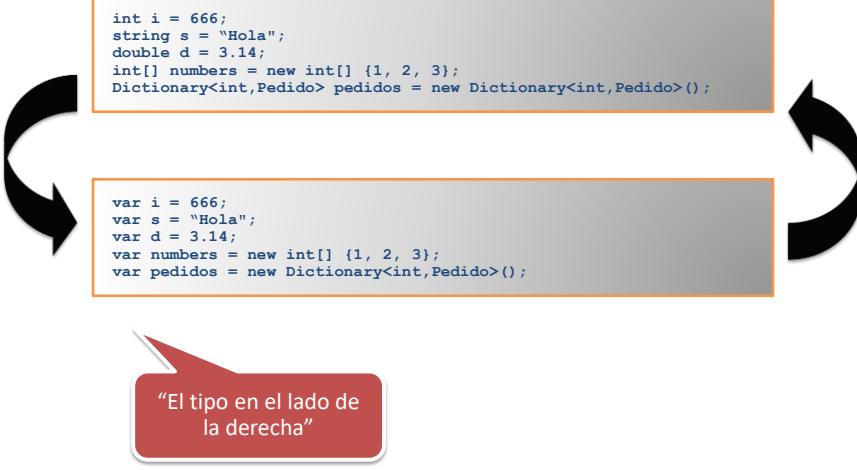
© JMA 2016. All rights reserved

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

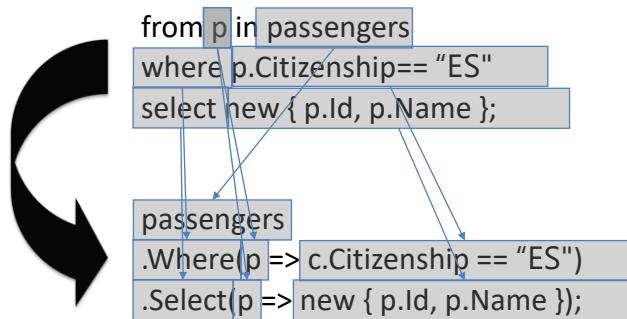
```
public class MiClase
{
    public static void Main() {
        List<Cliente> clientes = ObtenerListaClientes ();
        List<Cliente> locales =
            clientes.FindAll(c => c.Ciudad == "A Coruña");
    }
}
```

Expresión
Lambda

© JMA 2016. All rights reserved

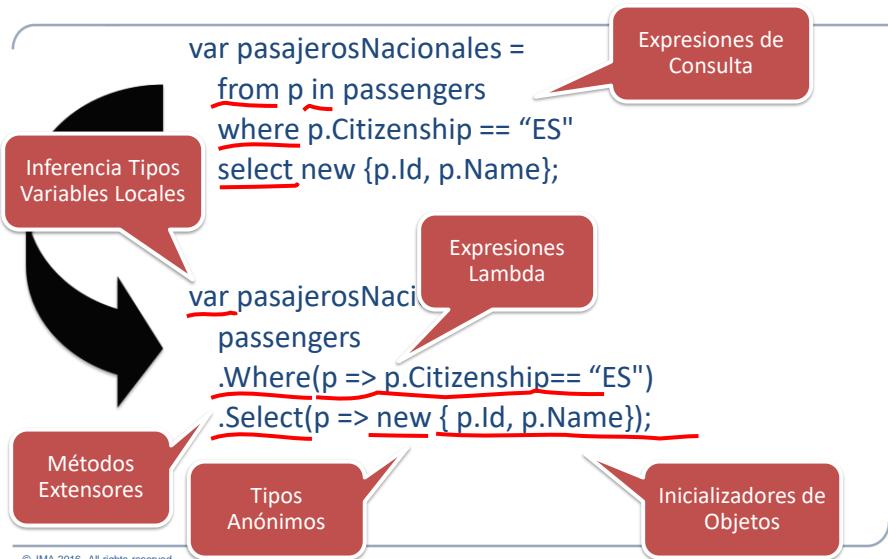
Introduciendo LINQ

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



© JMA 2016. All rights reserved

Introduciendo LINQ

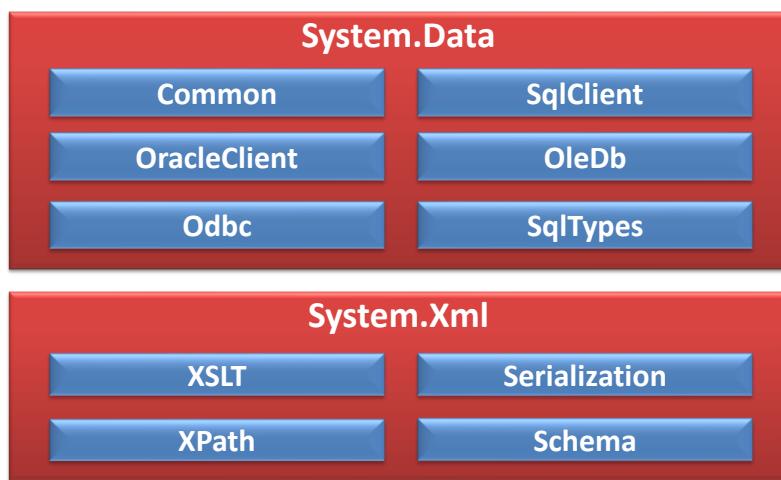


© JMA 2016. All rights reserved

ADO.NET

© JMA 2016. All rights reserved

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

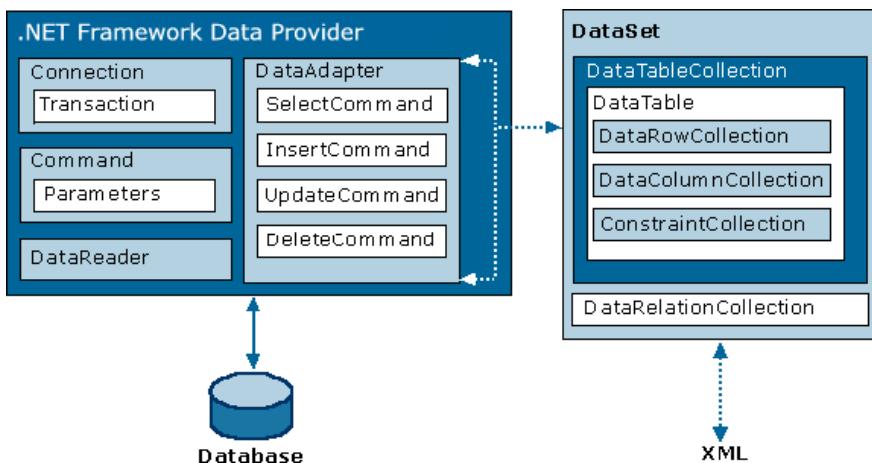
© JMA 2016. All rights reserved.

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



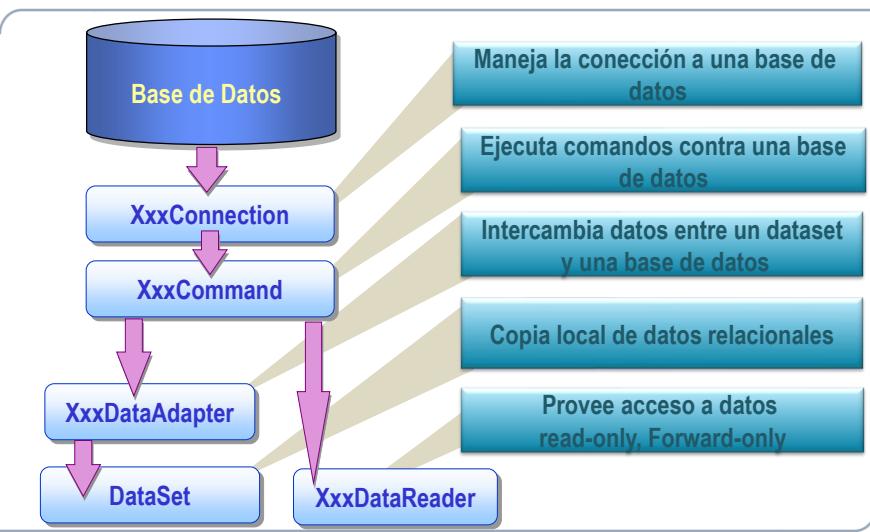
© JMA 2016. All rights reserved

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

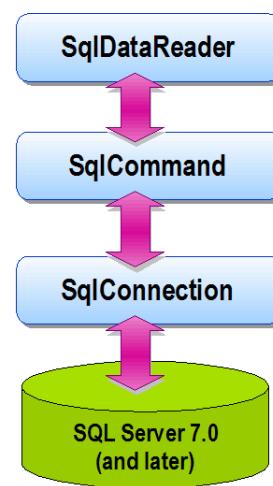
- **Connection:** Establece una conexión a un origen de datos determinado.
 - ConnectionString
 - Open, close, CreateCommand, BeginTransaction
- **Transaction:** Permite incluir comandos en las transacciones que se realizan en el origen de datos.
 - Commit, Rollback
- **Command:** Ejecuta un comando 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

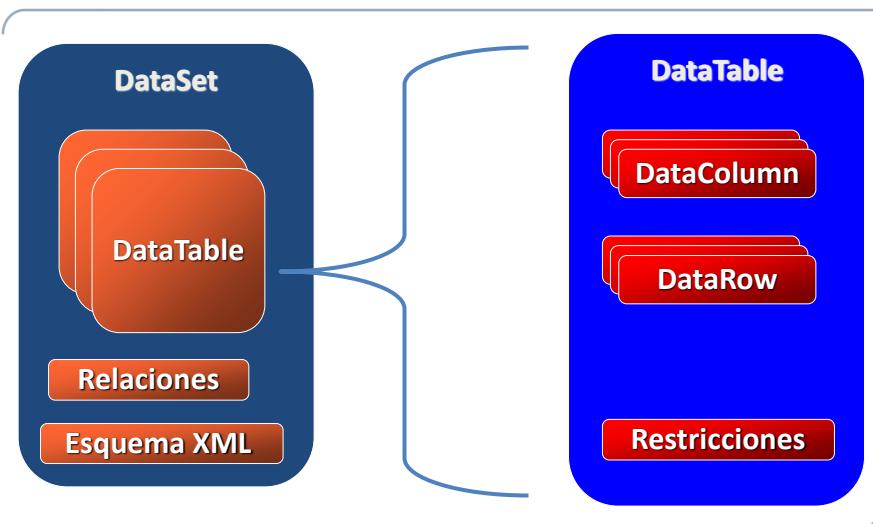
En un escenario conectado, los recursos se mantienen en el servidor hasta que la conexión se cierra:

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

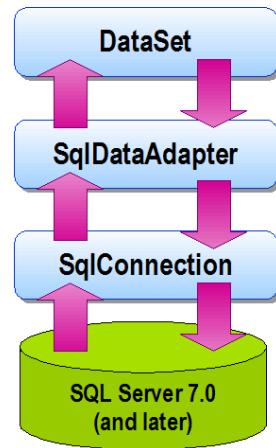
- **DataAdapter:** Llena un DataSet y realiza las actualizaciones necesarias en el origen de datos.
 - SelectCommand, InsertCommand, UpdateCommand y DeleteCommand
 - TableMappings, MissingMappingAction, MissingSchemaAction
 - AcceptChangesDuringFill, ContinueUpdateOnError
 - Fill, Update
 - Eventos: 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

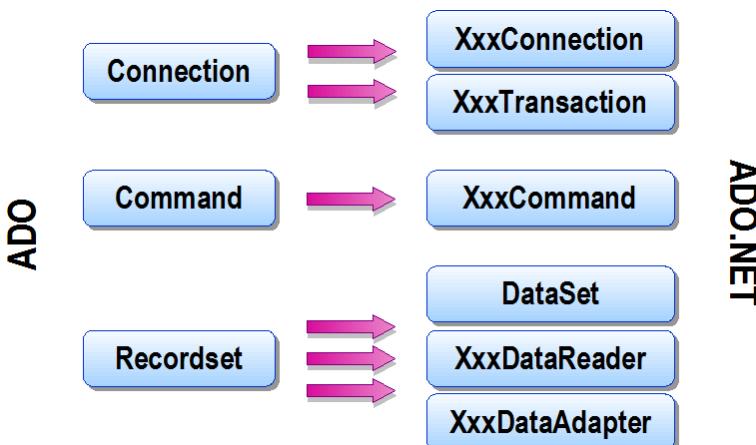
En un escenario desconectado, los recursos no se mantienen en el servidor mientras los datos se procesan:

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

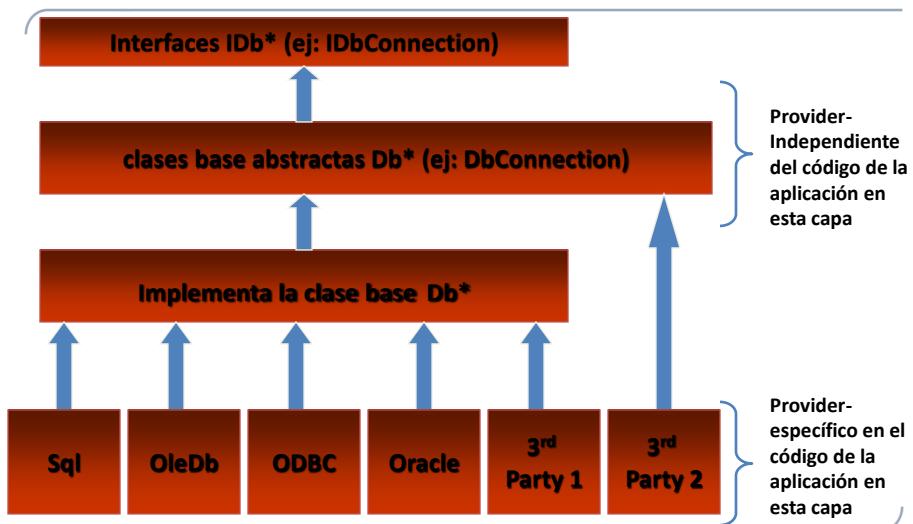
API Independiente

- 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)

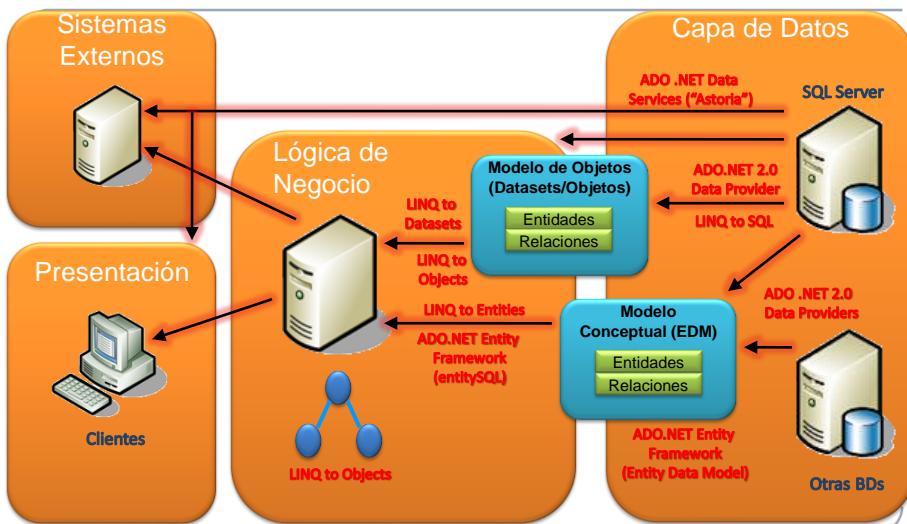
- Eleva el nivel de abstracción
- “Artefactos” reutilizables
- Crea un ecosistema de servicios

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

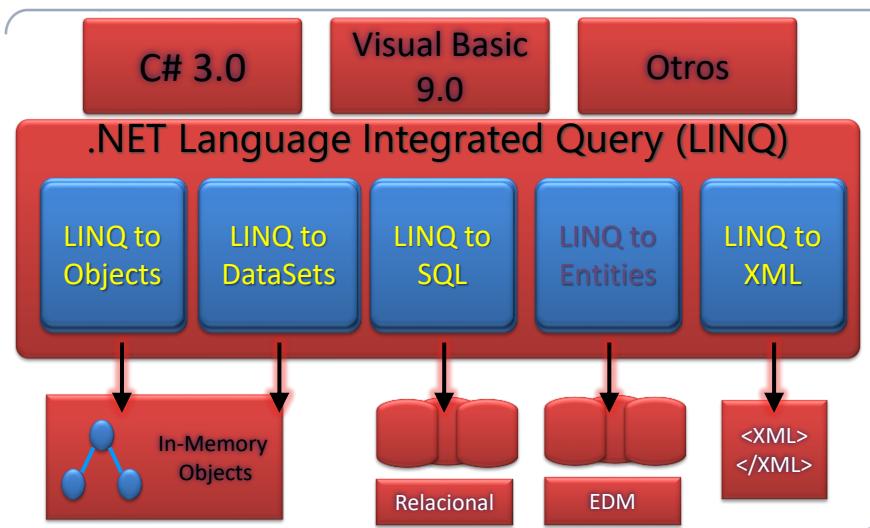
Language INtegrated Query

- 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



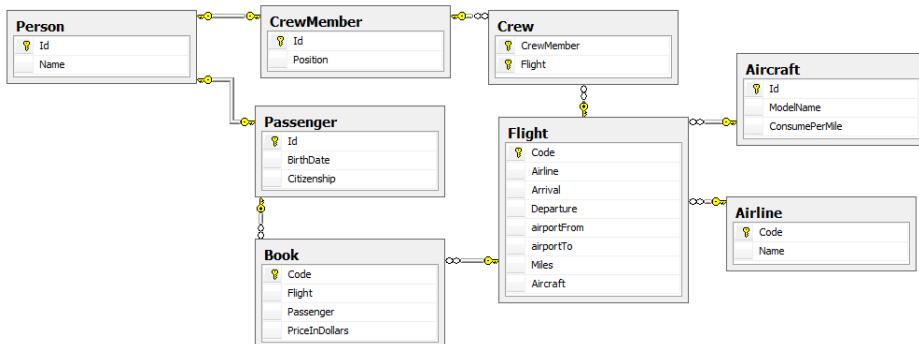
© JMA 2016. All rights reserved

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

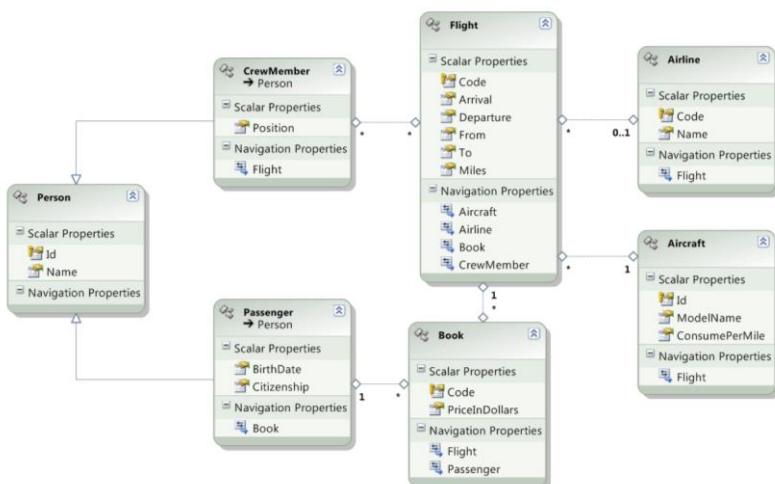
© JMA 2016. All rights reserved

Modelo entidad/relación



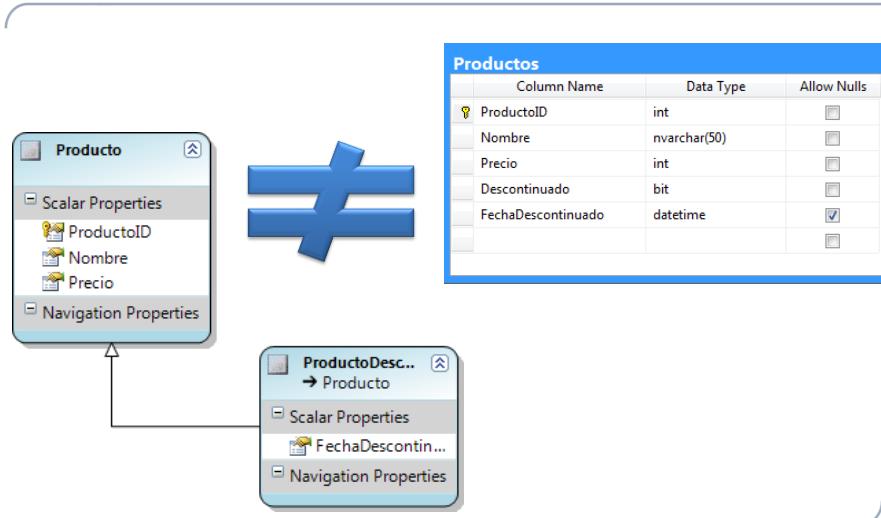
© JMA 2016. All rights reserved.

Modelo de dominio



© JMA 2016. All rights reserved.

Impedance Mismatch



© JMA 2016. All rights reserved.

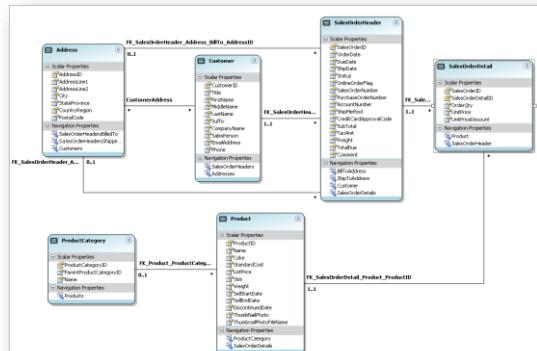
¿Por qué un ORM?

- 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 Data Model (EDM)

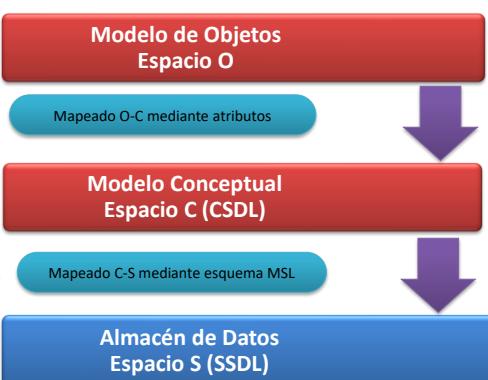
- Entity Data Model (EDM)
 - Vocabulario para describir el esquema del modelo conceptual
 - “Dibuja la aplicación que quieras ver”
- Entidades
 - Tipos distintos
 - Propiedades simples o complejas
- Relaciones
 - Describe las relaciones entre las entidades
 - Declaradas explícitamente: Nombre y cardinalidad



© JMA 2016. All rights reserved

Entity Data Model (EDM)

- Espacio S
 - Almacén o Storage
 - SSDL (Store Schema Definition Language)
 - Tablas, Vistas, SP...
 - Consulta igual que ADO.NET 2.0

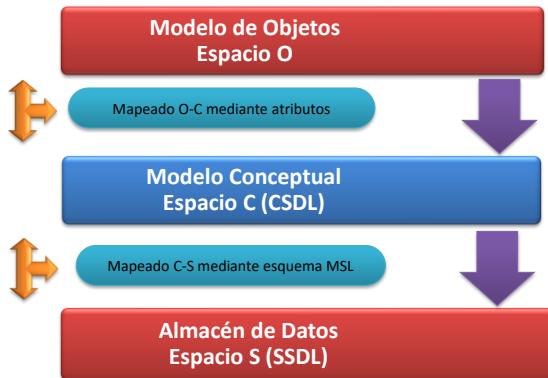


© JMA 2016. All rights reserved

Entity Data Model (EDM)

- Espacio C

- Modelo Conceptual
- CSDL (Conceptual Schema Definition Language)
- Entidades, Asociaciones
- Entity Data Provider
 - EntityConnection
 - EntityCommand
 - EntityDataReader
- E-SQL
- Necesitamos conocer:
 - SSDL
 - MSL (Mapeado entre SSDL y CSDL)



© JMA 2016. All rights reserved.

Entity Data Model (EDM)

- Espacio O

- Espacio superior
- Nivel de objeto en código.
- Mapeado con CSDL mediante atributos
- Con E-SQL
 - Para consultar usamos ObjectQuery<T>
- Con LINQ to Entities



© JMA 2016. All rights reserved.

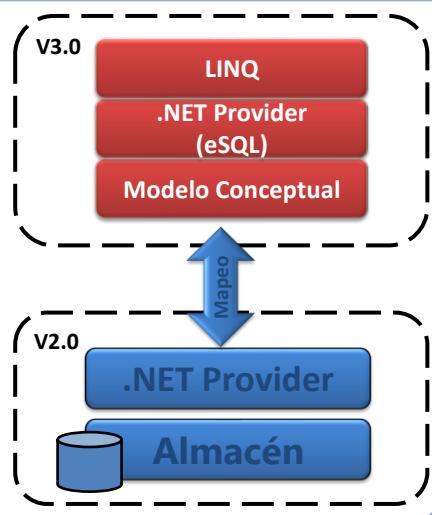
Entity Data Model (EDM)

Nivel	Se Modela Con	Contiene	Se Consulta Con
Espacio O	Código .NET	Clases, propiedades ...	- ObjectQuery<T> con E-SQL - Sentencias LINQ
Mapeado 1:1 Mediante atributos			
Espacio C	Conceptual Schema Definition Language	Entidades, Asociaciones.. Conjuntos	- EntityProvider utilizando E-SQL
Mapeado mediante MSL			
Espacio S	Store Schema Definition Language	Tablas, Vistas, procedimientos...	- ADO.NET con el proveedor específico

© JMA 2016. All rights reserved

LINQ to Entities

- La flexibilidad del modelado y mapeo del EF...con la productividad de LINQ



© 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

© JMA 2016. All rights reserved

Ventajas e Inconvenientes

- Ignorancia de la Persistencia (PI)
 - Complete Persistence Ignorance
 - POCO (Plain Old Code) – Atributos en Clases
 - IPOCO – Implementación de Interfaces
 - Clases Prescriptivas – Herencia de clases base
- EF – Actualmente IPOCO y Herencia
 - Mayor rendimiento y servicios base
- Futuro EF – Mayor independencia
 - POCO con metadatos externos
- Otros proveedores además de SQL Server

© JMA 2016. All rights reserved

Ventajas para el desarrollador

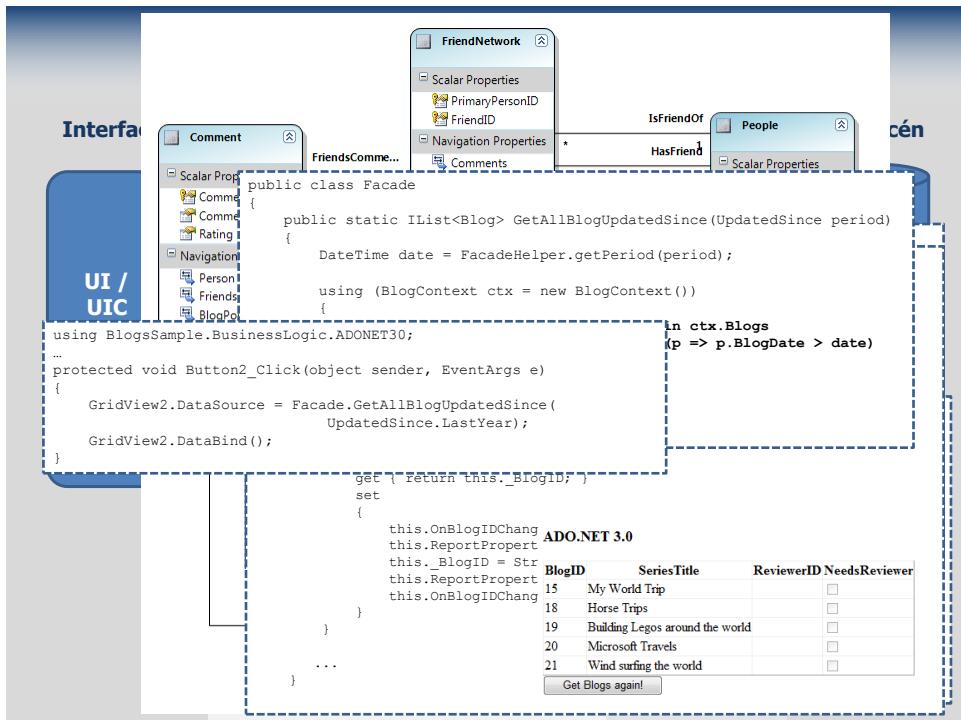
- Alternativas a TSQL
 - LINQ to Entities
 - Entity SQL
- Reducción de líneas de código de la capa de datos
- Capacidad de crear la BD a partir del modelo
- Mayor seguridad frente a SQL Injections
- Abstracción total de la Base de datos
 - Servidor de base de datos
 - SQL Server
 - Oracle
 - MySQL

© JMA 2016. All rights reserved

Arquitectura de EF

- OR/M + Objetos de Servicio
 - Soporte de varios SGDB: EntityClientProvider
 - Herramientas y lenguaje para mapeado
 - Modelo Conceptual: EntityObject
 - Contextos tipados: ObjectContext
 - Gestión de Entidades: ObjectStateManager
 - Consulta: eSQL y LINQ To Entities

© JMA 2016. All rights reserved



Modelo Conceptual en EF

- Clases prescriptivas
 - Structural Object > EntityObject
 - Gestión de identidad (EntityKey)
 - Gestión de cambios (TrackingEntity event)
 - Soporte para relaciones (EntityCollection)
 - Estado (EntityState)
 - Son clases parciales
- Posibilidad de clases IPOCO
 - Implementar IEntityWithKey, IEntityWithChangeTracker, IEntityWithRelationship

Object Context

- Clase derivada (generada) de ObjectContext
- Tipado Fuerte: Manipulación del Modelo Conceptual
 - Consultas/Recuperación: Blogs: ObjectQuery;
 - Inserciones: .AddToBlog(Blog b); .AddObject(...),
 - Borrado: .DeleteObject
 - Persistencia: .SaveChanges();
- Gestión de la conexión
- Metadata (a partir de CSDL)
- Almacén o caché en memoria de objetos
- Tracking de estado objetos:
 - .Attach(..), .Detach(..)
 - ObjectStateManager
 - MergeOption

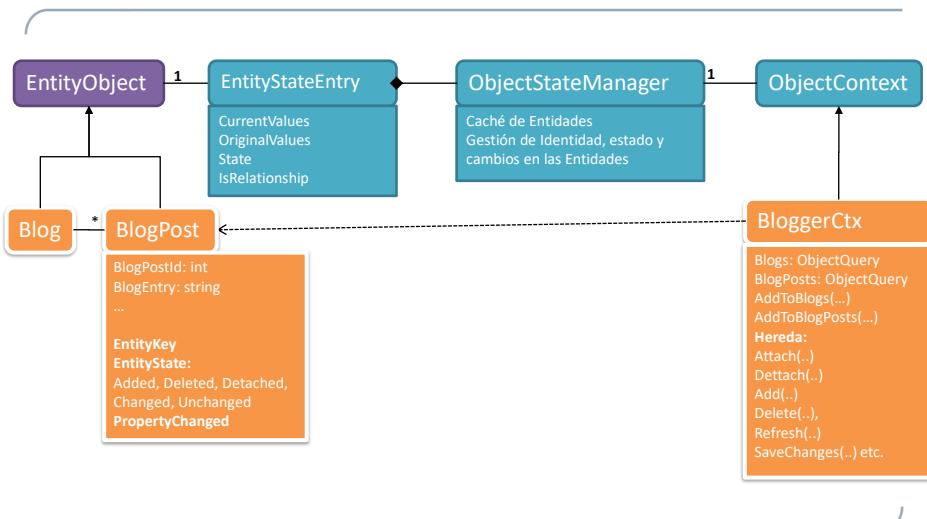
© JMA 2016. All rights reserved

ObjectStateManager

- Seguimiento del estado de entidades
- Gestiona entradas EntityStateEntry para cada Entidad en almacen en memoria.
 - Cuando se cargan (Query, Attach): Unchanged
 - Cuando se crean (AddObject): Added
 - Cuando se modifican: Changed
 - Cuando se borran: Deleted
 - Cuando se destruye el ObjectContext: Detached
 - Al aplicar ObjectContext.SaveChanges() en Added, Changed, cambia a Unchanged.

© JMA 2016. All rights reserved

Diseño: Entity Framework



© JMA 2016. All rights reserved

Consultas

- **Entity SQL**
 - Dialecto SQL indep. de gestor con soporte para:
 - Tipos enriquecidos, relaciones, herencia...
 - Strings que se resuelven en tiempo de ejecución
- ```
ObjectQuery<Blog> query = ctx.CreateQuery<Blog>(
 "SELECT VALUE bp.Blogs FROM BlogPosts as bp WHERE bp.BlogDate > @date",
 new ObjectParameter("date", date));
```
- **LINQ to Entities**
    - Todas las ventajas de LINQ (tipado fuerte, expresiones lambda)
    - Lenguaje que se resuelve en tiempo de compilación
    - Aprovechamos el tipado fuerte, la inferencia y el Intellisense de Visual Studio
    - Menos errores en ejecución
- ```
IQueryable<Blog> query = from posts in ctx.BlogPosts
                           where posts.BlogDate > date
                           select posts.Blogs;
```

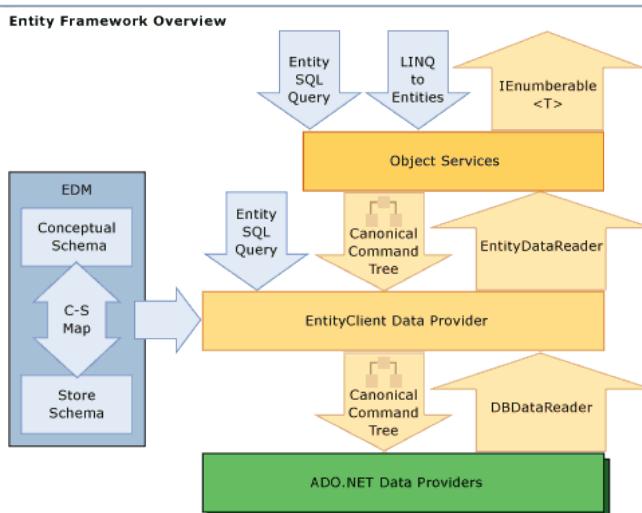
© JMA 2016. All rights reserved

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

Arquitectura de LINQ To Entities



© JMA 2016. All rights reserved

Operadores disponibles

Expresión de consulta de Linq	Where(), Select(), SelectMany(), OrderBy(), ThenBy(), OrderByDescending(), ThenByDescending(), GroupBy(), Join(), GroupJoin()
Partición	Take(), Skip()
Conjunto	Distinct(), Union(), Intersect(), Except()
Conversión	ToArray(), ToList(), ToDictionary(), ToLookup(), AsEnumerable(), Cast<T>(), OfType<T>()
Generación	N/A
Cuantificación	Any(), All()
Elementos	First(), Last(), ElementAt(), {método}OrDefault()
Agregados	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

Otras consideraciones

- Pensar la consulta con el modelo conceptual
 - Navegar por las relaciones y no con joins
- Los objetos recuperados dentro del contexto son gestionados por el ObjectStateManager (tracking)
- Cuando se cierra el contexto NO.
- Carga Perezosa de relaciones (dentro de un contexto)
 - ¡¡No nos lo vamos a traer todo!!
 - Si en el where se utiliza una relación, ésta se carga
 - Carga implícita
 - from f in ctx.Flights.Include("Aircraft") select f;
 - Carga explícita
 - if (!flight.Aircraft.IsLoaded)
aflight.Aircraft.Load();

© JMA 2016. All rights reserved

DATOS

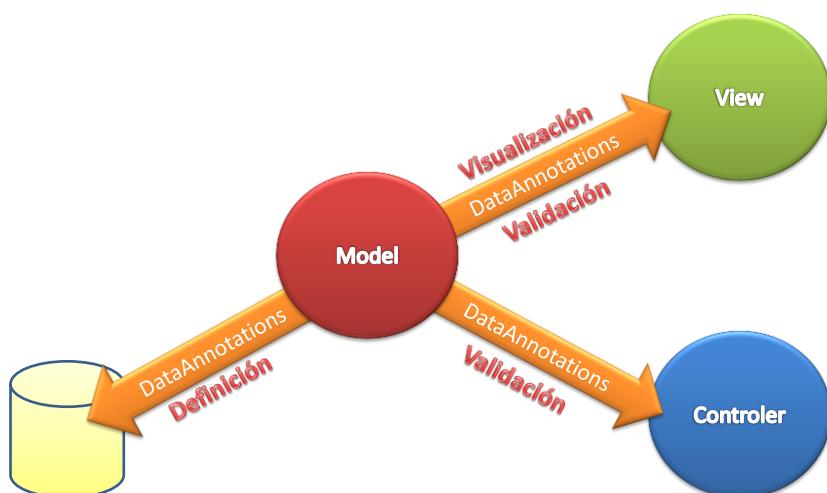
© JMA 2016. All rights reserved

DataAnnotations

- System.ComponentModel.DataAnnotations;
- Permiten definir los metadatos de los modelos de datos para su uso por entidades externas.
- 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.
System.Web.Mvc	
HiddenInput	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 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.
- Interfaces relacionados: IDataErrorInfo, INotifyDataErrorInfo

© JMA 2016. All rights reserved

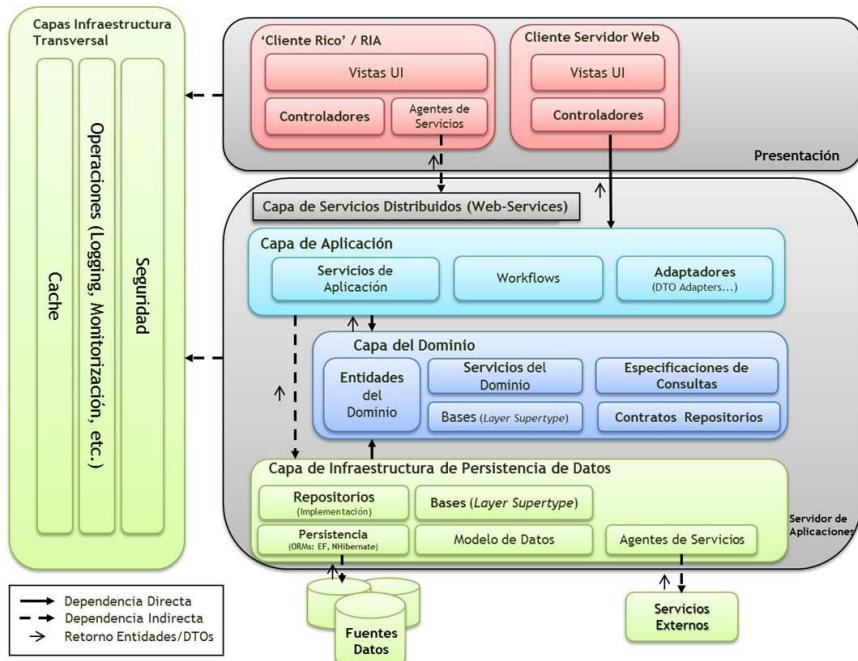
Anotar clases ya existentes

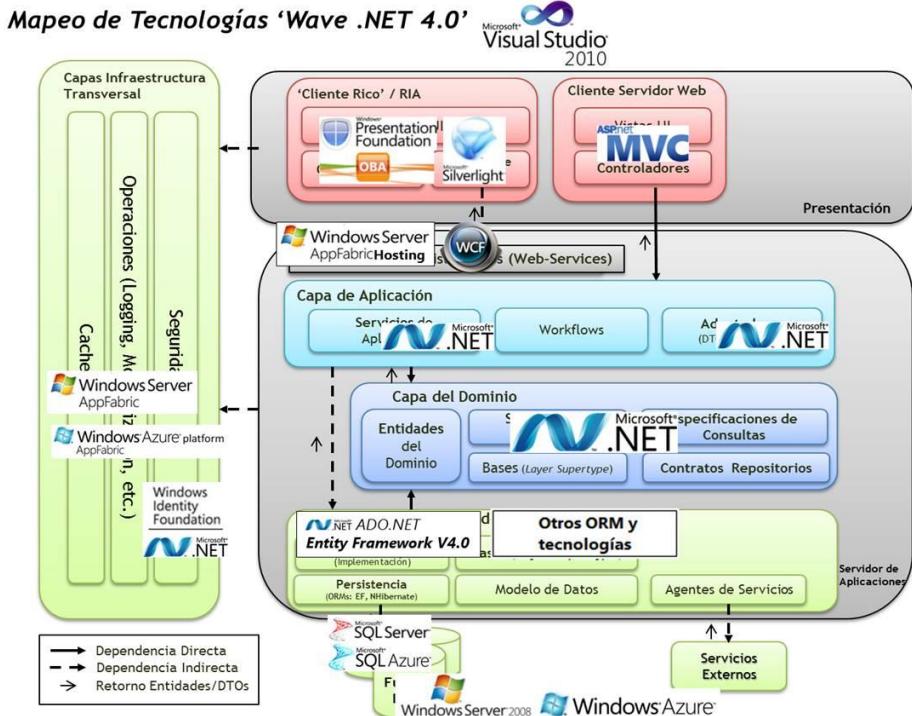
- Se requiere una clase auxiliar con las propiedades a decorar decoradas.
- Declarativa: [MetadataType(typeof(tipo))]
– Se aplica con una partial class (en el mismo ensamblado que la clase a anotar).
- Imperativa: System.ComponentModel.TypeDescriptor y AssociatedMetadataTypeTypeDescriptionProvider:

```
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





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 pertenece 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).

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 asincrónica

- Patrón asincrónico basado en tareas (TAP) , que utiliza un método único para representar el inicio y la finalización de una operación asincrónica. TAP Se presentó en .NET Framework 4. Es el enfoque recomendado para la programación asincrónica 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 asincrónico basado en eventos (EAP), que es el patrón heredado basado en eventos para proporcionar el comportamiento asincrónico. 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 asincrónica (APM) (también denominado `IAsyncResult`), que es el modelo heredado que usa la interfaz `IAsyncResult` para ofrecer un comportamiento asincrónico. En este patrón, las operaciones sincrónicas requieren los métodos `Begin` y `End` (por ejemplo, `BeginWrite` y `EndWrite` para implementar una operación de escritura asincrónica). Este patrón ya no se recomienda para nuevo desarrollo.

© JMA 2016. All rights reserved

Modelos de diseño asíncronos

- Una operación asincrónica se ejecuta en un subproceso independiente del subproceso de aplicación principal. Cuando una aplicación llama a métodos para que realicen una operación de forma asincrónica, la aplicación puede seguir ejecutándose mientras el método asincrónico efectúa su tarea.
- Programación asincrónica mediante `IAsyncResult`
 - Métodos:
 - Método Síncrono: `NombreDeOperación`
 - Métodos Asíncronos: `BeginNombreDeOperación` y `EndNombreDeOperación`
- Programación asincrónica mediante delegados
 - Métodos:
 - Método Síncrono: `Invoke`
 - Métodos Asíncronos: `BeginInvoke` y `EndInvoke`
- Programación asincrónica 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
Mutex	Primitiva de sincronización que puede utilizarse también para la sincronización entre procesos.
Monitor	Proporciona un mecanismo que sincroniza el acceso a los objetos.
Interlocked	Proporciona operaciones atómicas para las variables compartidas por varios subprocessos.
ReaderWriterLock	Define un bloqueo que admite un escritor y varios lectores.
Semaphore	LIMITA el número de subprocessos que pueden tener acceso a un recurso o grupo de recursos simultáneamente.
ThreadPool	Proporciona un grupo de subprocessos que pueden utilizarse para exponer elementos de trabajo, procesar la E/S asincrónica, 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 {0:0}!", 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 asincrónicos (ver. 5.0)

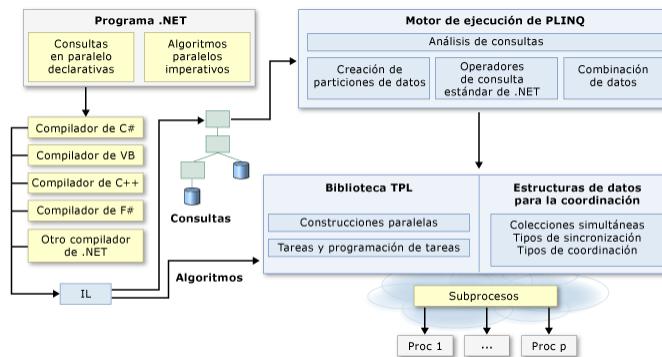
- Las palabras clave `async` y `await` permiten crear un método asincrónico casi tan fácilmente como se crea un método sincrónico.

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

© JMA 2016. All rights reserved

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