educación IT
CODE YOUR FUTURE

Introducción a C# .NET

Módulo 1

Implementaciones de .NET

¿Qué es .NET?

Es un amplio conjunto de bibliotecas de desarrollo **multilenguaje y multiplataforma** que está pensado para correr sobre distintos entornos operativos, y que pretende unificar los distintos paradigmas de la programación (escritorio, cliente-servidor, WEB, APIs, Datos, Business Intelligence, etc.) en un mismo ambiente de trabajo.

.NET está formado por **cuatro componentes principales**, a saber:

1. **El CLR (*Common Language Runtime*)**: es el entorno de ejecución para las aplicaciones codificadas en alguno de los diferentes lenguajes que cumplen con los requisitos sintácticos y semánticos de .NET.

2. **El MSIL (*Microsoft Intermediate Language*)**: representa el *código objeto* al que es transformado el *código fuente* escrito en cualquiera de los lenguajes aceptados por .NET. Este código fuente luego va a ser compilado a código nativo por el compilador **JIT** (*Just In Time*) que provee el CLR en tiempo de ejecución.

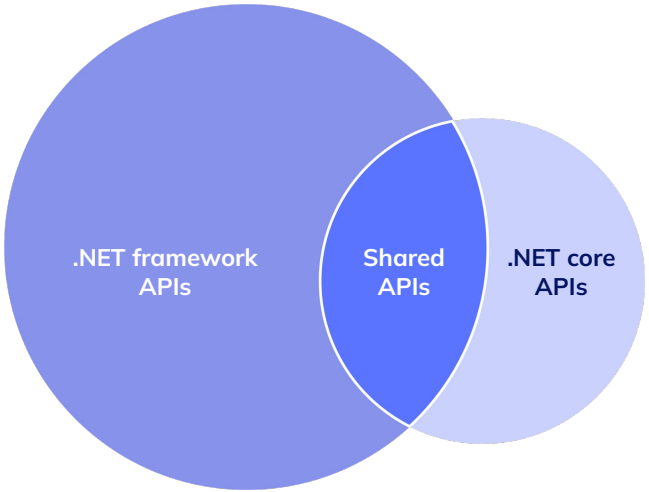
3. **La biblioteca de clases .NET *Framework***: es una extensísima colección de *clases base* (plantillas), funcionales para todos los propósitos imaginables, que es implementada y usada por todos los lenguajes e IDE's que corren sobre .NET.

4. **El entorno de programación (IDE)**: es el ambiente de trabajo propiamente dicho conformado por herramientas de diseño, editores, depuradores y diversos asistentes, para que el programador pueda diseñar, codificar y testear sus aplicaciones. Existen varios (Sharp Develop, RAD Studio, etc.), pero nosotros en este curso y posteriores usaremos el propio de Microsoft: Visual Studio Code.

Implementaciones de .NET

En un sentido más técnico y no tan orientado a los lenguajes de programación, una aplicación de .NET se desarrolla y se ejecuta en una o varias *implementaciones de .NET*. Las implementaciones de .NET incluyen los siguientes componentes: **.NET Framework**, **.NET Core** y **Mono**.

Hay una especificación de API (*Application Programming Interface*, un conjunto de librerías de código), común a todas las implementaciones de .NET denominada **.NET Standard**. Veremos una breve introducción a cada uno de estos conceptos.



.NET Standard

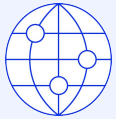
.NET Standard es un conjunto de APIs que se implementa mediante la biblioteca de **clases base** de una implementación de .NET.

Más formalmente, es una especificación de APIs de .NET que constituye 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; por ejemplo, esto logra que la implementación .NET Core sea multiplataforma.

.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.

DESKTOP

WPF
Windows Forms
UWP



WEB

ASP.NET

CLOUD

Azure

MOBILE

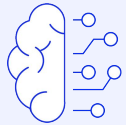
Xamarin

GAMING

Unity

IoT

ARM32
ARM64



AI

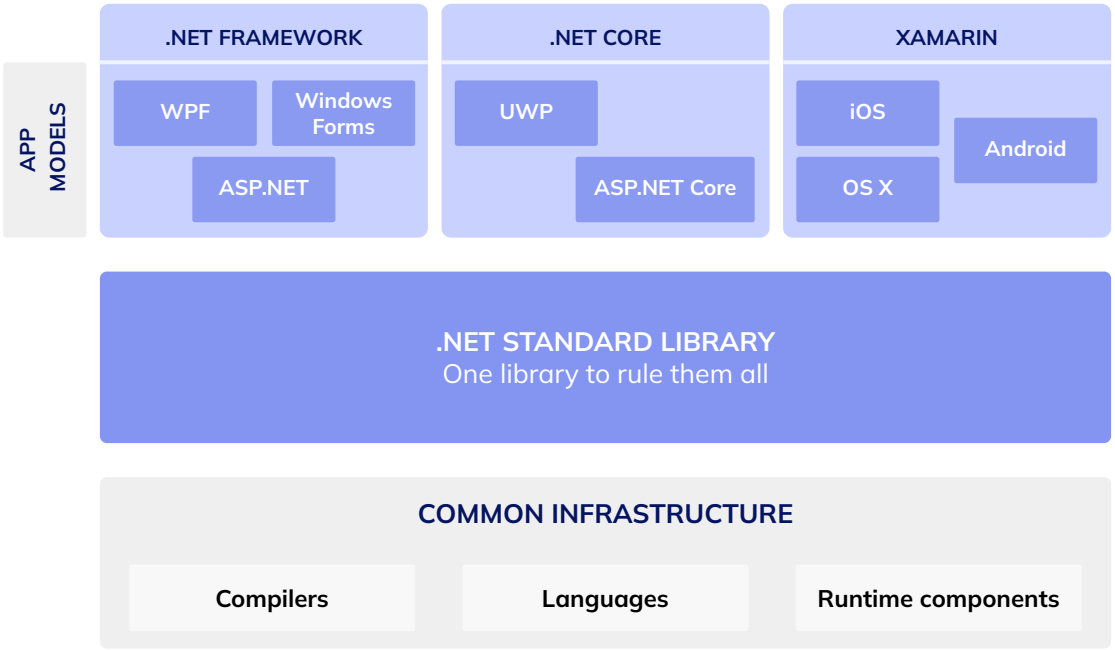
ML.NET
.NET for
Apache Spark

.NET FRAMEWORK	.NET CORE	XAMARIN
WINDOWS APPLICATIONS	CROSS-PLATFORM SERVICES	MOBILE APPLICATIONS

.NET STANDARD

INFRASTRUCTURE

RUNTIME COMPONENTS	COMPILERS	LANGUAGES


APP MODELS
.NET FRAMEWORK
WPF
Windows Forms
ASP.NET
.NET CORE
UWP
ASP.NET Core
XAMARIN
iOS
Android
OS X
.NET STANDARD LIBRARY
One library to rule them all
COMMON INFRASTRUCTURE
Compilers
Languages
Runtime components

Implementaciones de .NET

Cada implementación de .NET incluye los siguientes componentes (luego se profundizará conceptualmente en algunos de ellos):

Uno o varios **entornos de ejecución,** por ejemplo:

- **CLR** para .NET Framework (Common Language Runtime)
- **CoreCLR** y **CoreRT** para .NET Core (.NET Core Common Language Runtime y .NET Core Runtime)

Una **biblioteca de clases** que implementa .NET Standard y puede implementar API adicionales, por ejemplo:

- Biblioteca de clases base de .NET Framework
- Biblioteca de clases base de .NET Core (.NET Core Base ClassLlibrary)

Opcionalmente, uno o varios **marcos de trabajo** de la aplicación, por ejemplo:

- ASP.NET para .NET Framework y para .NET Core.
- Windows Forms para .NET Framework.
- Windows Presentation Foundation (WPF) para .NET Framework.

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 Core, .NET Framework, Mono y UWP.**

.NET Core

.NET Core es una implementación multiplataforma de .NET diseñada para aplicaciones instaladas tanto en servidores propios como servidores en la nube a gran escala.

.NET Core puede ejecutarse en entornos Windows, macOS y Linux. Implementa .NET Standard, de forma que cualquier código que tenga como destino .NET Standard se pueda ejecutar en .NET Core. ASP.NET Core se ejecuta en .NET Core.

.NET Framework

.NET Framework es la implementación original de .NET que existe desde 2002. Es el mismo entorno que los desarrolladores de .NET han usado siempre. 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.

Contiene APIs específicas de Windows adicionales, como la API para el desarrollo de aplicaciones de escritorio con Windows Forms y WPF. .NET Framework está optimizado para crear aplicaciones de escritorio de Windows.

¿Qué es C#?

C# (se pronuncia "*se sharp*") es un lenguaje que pretende reunir lo mejor de los diversos lenguajes que compilan a nativo (C / C++, Object Pascal, ADA, etc.) y de los interpretados (Java, Perl, etc.) en uno solo, y que, además, pueda correr sobre diversos sistemas operativos.

Al respecto, existe un proyecto paralelo al desarrollo de .NET de Microsoft, que se denomina "Mono" que transportó el lenguaje C# y gran parte del framework .NET a Unix / Linux / Solaris.

C# toma gran parte de la sintaxis de C / C++ y muchas de las características de Eiffel (un lenguaje de laboratorio que ideó Bertrand Meyer), ofreciendo al desarrollador un potente **lenguaje 100% orientado a objetos.**

Si bien el CLR acepta muchos otros lenguajes, el framework .NET está 100% escrito en C#; por lo tanto, ***es el lenguaje base para .Net***.

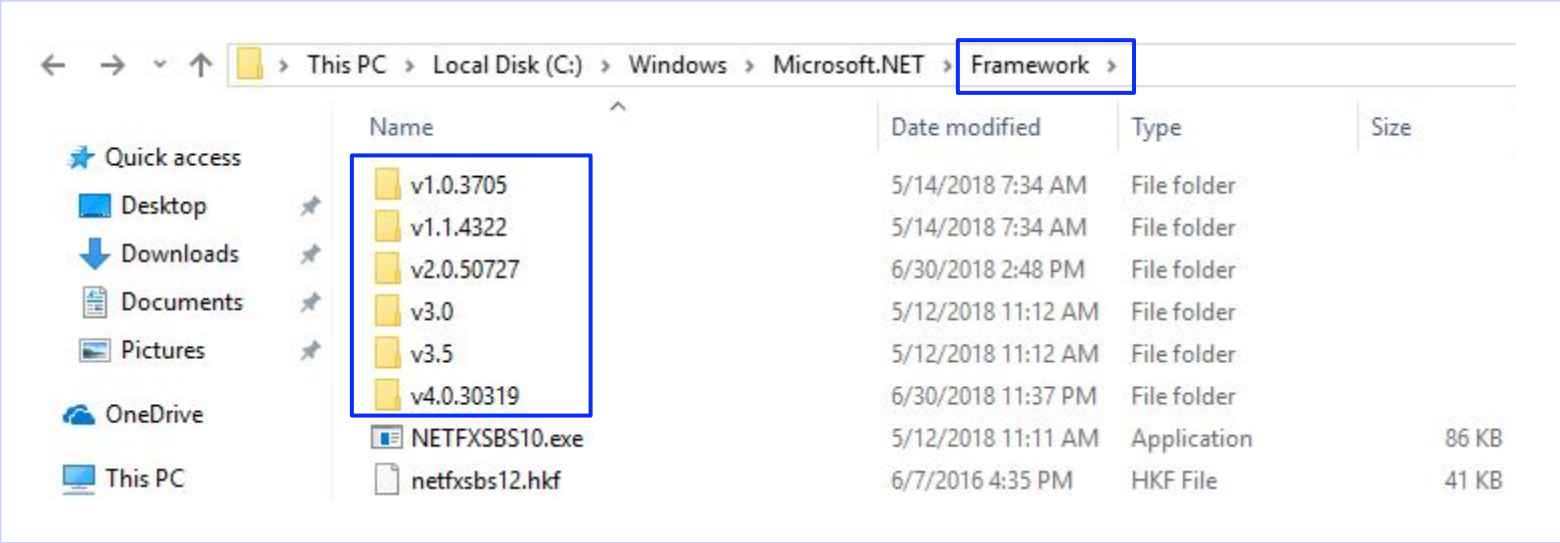
Entorno de desarrollo

El compilador

Cuando se instala en cada computadora el SDK de Microsoft .NET, se crea el directorio ***C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\..*** , dentro del cual se alojan los compiladores de línea.

El compilador para C# es "***csc.exe***" y puede ser invocado desde la línea de comandos, es decir: **Inicio | Ejecutar | cmd**.

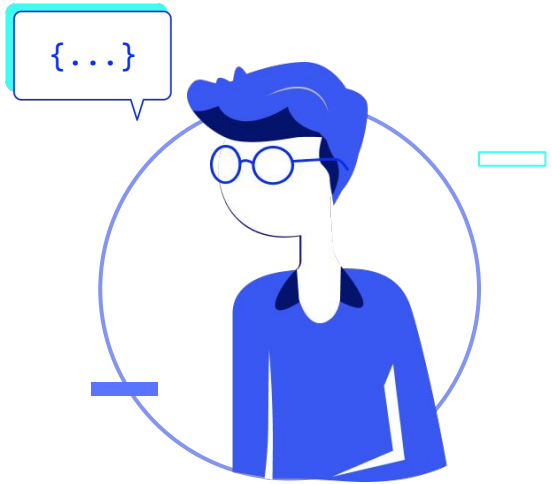
Cuando se invoca al compilador pasándole como parámetro un nombre de archivo cualquiera con la extensión "*.cs*", convierte a este en un archivo **.exe* (ejecutable), pero este **NO** es un ejecutable NATIVO (es decir, autónomo), sino que necesita del framework .NET con el CLR instalado en la misma máquina para poder correr.


This PC
Local Disk (C:)
Windows
Microsoft.NET
Framework
Quick access
Desktop
Downloads
Documents
Pictures
OneDrive
This PC
Name
Date modified
Type
Size
v1.0.3705
5/14/2018 7:34 AM
File folder
v1.1.4322
5/14/2018 7:34 AM
File folder
v2.0.50727
6/30/2018 2:48 PM
File folder
v3.0
5/12/2018 11:12 AM
File folder
v3.5
5/12/2018 11:12 AM
File folder
v4.0.30319
6/30/2018 11:37 PM
File folder
NETFXSBS10.exe
5/12/2018 11:11 AM
Application
86 KB
netfxsbs12.hkf
6/7/2016 4:35 PM
HKF File
41 KB

Los archivos fuente (“*.cs”)

Los archivos de “código fuente” no son más que **archivos de texto plano,** que contienen las líneas del programa en C# con las instrucciones que cada programador quiere que la aplicación lleve a cabo. Deben cumplir con las normas sintácticas del lenguaje, caso contrario, el compilador detecta los errores y no genera el ejecutable.

Veamos un primer ejemplo de aplicación de consola en la siguiente slide.

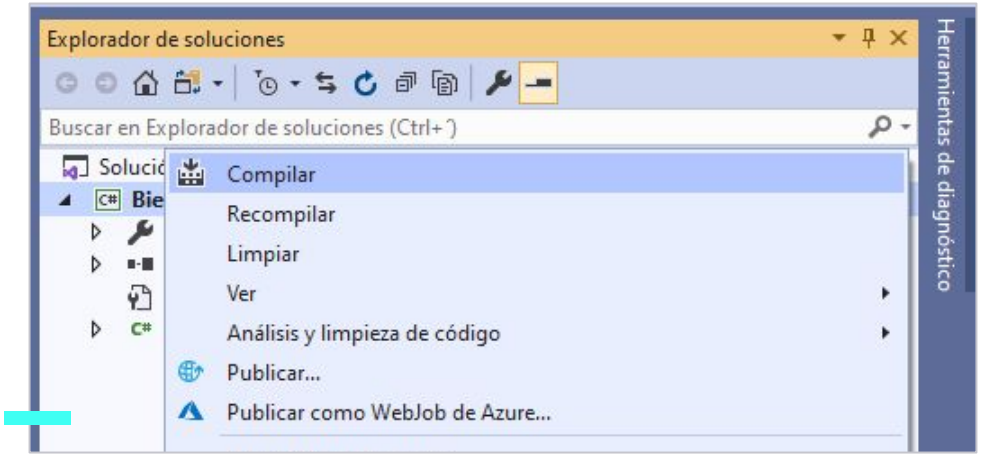


```
/*
 Nombre del archivo: Saludo.cs
 Autor: ...
 Objetivo del programa: Mostrar por consola un mensaje de bienvenida
*/

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

...

```
// Nombre del archivo ejecutable (mismo nombre que el Proyecto) .exe que se generará
namespace Bienvenida
{
// Clase inicial del programa Saludo.cs
class Saludo
    {
// Punto de entrada a la aplicación (el código dentro de Main es lo primero en ejecutarse)
static void Main()
        {
      // Muestra por pantalla el contenido entre "" dentro del paréntesis
      Console.WriteLine("Hola a todos !!!Bienvenidos a .Net !!!");

      // Espera que el usuario presione una tecla para continuar la ejecución del programa
      Console.ReadKey();
        }
    }
}
```

En la imagen anterior codificamos el programa en el IDE de Visual Studio, y aquí también podríamos compilarlo:



Cómo está formado el framework .NET

La arquitectura del framework .NET, es la **implementación del CLI (*Common Language Runtime*)** por parte de Microsoft para los lenguajes C#, Visual Basic, J#, ASP, y JavaScript, más varios paquetes que dan soporte a **interfaces de usuario, acceso a datos, XML y WEB agrupadas en una *Librería de Clase Base* (BCL)** que está formada por cientos de tipos de datos.

Esta librería está **escrita en MSIL,** por lo que puede usarse desde cualquier lenguaje cuyo compilador genere MSIL.

A través de las clases suministradas en ella es posible desarrollar cualquier tipo de aplicación, desde las tradicionales aplicaciones de ventanas, consola o servicio de Windows NT hasta los novedosos servicios Web y páginas ASP.NET.

Dada su amplitud, ha sido necesario organizar las clases, incluidas en ella, en ***espacios de nombres*** que agrupan clases con funcionalidades similares. Esto es una ***clasificación.***

Veamos los espacios de nombres más usados en la tabla del próximo slide.

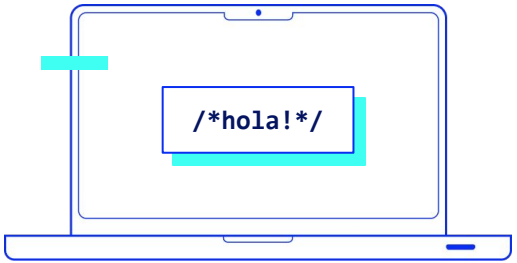
Espacio de nombres	Utilidad de los tipos de dato que contiene
System	Tipos muy frecuentemente usados, como los tipos básicos, tablas, excepciones, fechas, números aleatorios, recolector de basura, entrada/salida en consola, etc.
System.Collections	Colecciones de datos de uso común como pilas, colas, listas, diccionarios, etc.
System.Data	Manipulación de bases de datos. Forman la denominada arquitectura ADO.NET.
System.IO	Manipulación de archivos y otros flujos de datos.
System.Net	Realización de comunicaciones en red.
System.Reflection	Acceso a los metadatos que acompañan a los módulos de código.
System.Runtime.Remoting	Acceso a objetos remotos.
System.Security	Acceso a la política de seguridad en que se basa el CLR.
System.Threading	Manipulación de hilos de ejecución.
System.Web.UI.WebControls	Creación de interfaces de usuario basadas en ventanas para aplicaciones Web.
System.Windows.Forms	Creación de interfaces de usuario basadas en ventanas para aplicaciones estándar.
System.XML	Acceso a datos en formato XML.

Comentarios

¿Qué es un comentario?

Un **comentario** es texto que se incluye en el código fuente para facilitar su lectura a los programadores y cuyo contenido es, por omisión, completamente ignorado por el compilador.

En C# hay dos formas de escribir comentarios: la primera consiste en encerrar todo el texto que se desee comentar entre los caracteres `/*` y `*/` con la siguiente sintaxis: `/*<texto>*/`



Utilización de comentarios

Estos comentarios pueden abarcar tantas líneas como sea necesario. Por ejemplo:

```
/*
Esto es un comentario
que muestra cómo se comentan varias líneas
*/
```

No es posible anidar comentarios de este tipo.
Es decir, no vale escribir comentarios como el siguiente:

```
/*  Comentario contenedor /* Comentario contenido */  */
```

Dado que muchas veces los comentarios que se escriben son muy cortos y no suelen ocupar más de una línea, C# ofrece una **sintaxis alternativa más compacta** para la escritura de este tipo de comentarios, en las que se considera como indicador del comienzo del comentario los caracteres **//** y como indicador de su final el fin de línea.

Por tanto, la sintaxis que siguen estos comentarios es: **// <texto>**

Y un ejemplo de su uso es:

```
// Este es un comentario de una sola línea
```

Estos comentarios de una sola línea **sí pueden anidarse** sin ningún problema. Por ejemplo, el siguiente comentario es perfectamente válido:

```
// Comentario contenedor // Comentario contenido
```

Además, el IDE de **Visual Studio** tiene en botones dos acciones: una para **comentar** un bloque de instrucciones y otra para **"descomentarlo",** resultando en muchas ocasiones más ágiles.

educación IT
CODE YOUR FUTURE

¡Sigamos trabajando!