- Antonio Pelleriti -

PROGRAMMARECON

C# 7

Guida completa

Compilatore e ambiente di sviluppo Visual Studio 2017 >>

La programmazione a oggetti, eventi, eccezioni, generics >>

Sviluppo per Windows, Linux e macOS >>

Sintassi e costrutti del linguaggio >>

*pro DigitalLifeStyle

*pro
Digital*LifeStyle

Programmare con C# 7

Guida completa

Antonio Pelleriti

EDIZIONI
LSWR

Programmare con C# 7 | Guida completa

Autore: Antonio Pelleriti

Collana: *pro DigitalLifeStyle

Publisher: Marco Aleotti
Progetto grafico: Roberta Venturieri
Immagine di copertina: © izabell | Thinkstock

ISBN: 978-88-6895-577-9

EDIZIONI LSWR

Via G. Spadolini, 7
20141 Milano (MI)
Tel. 02 881841
www.edizionilswr.it

Printed in Italy

Finito di stampare nel mese di settembre 2017 presso "Rotomail Italia" S.p.A., Vignate (MI)

(*) Edizioni Lswr è un marchio di La Tribuna Srl. La Tribuna Srl fa parte di LSWR GROUP.

Sommario

Introduzione

C# è un linguaggio di programmazione sviluppato da Microsoft, all'interno della piattaforma **.NET**, e divenuto ormai uno dei protagonisti principali sul palcoscenico dello sviluppo software, avendo fatto il suo debutto nel 2000, e continuando a mantenersi sempre in linea con le tendenze, versione dopo versione, con l'aggiunta di caratteristiche e funzionalità sempre nuove, tanto da essere, oggi più che mai, costantemente ai primi posti nelle graduatorie dei linguaggi più scelti e utilizzati dalla comunità mondiale di sviluppatori, professionali, hobbisti, o di chi si affaccia per la prima volta a tale mondo.
In questa sua evoluzione, il linguaggio non è mai divenuto complesso o pesante da digerire, e anzi ha mantenuto la semplicità e contemporaneamente la potenza che gli permettono di affrontare e risolvere problemi legati ad ambiti di sviluppo eterogenei, sia dal punto di vista della piattaforma di esecuzione che da quello prettamente pratico e legato all'ambito applicativo.
C# e .NET rappresentano oggi una delle principali scelte per chi vuole creare software che giri sul desktop di un personal computer, come applicazione lato server, oppure applicazione web all'interno di un browser internet, o ancora come app installata su uno smartphone o su un tablet, spaziando in qualunque caso lungo **scenari applicativi** anch'essi estremamente eterogenei: dal mondo dell'industria a quello dei software di produttività e gestione aziendale, passando per i videogame e per i sistemi di commercio elettronico.
L'introduzione di .NET ha costituito senza dubbio una delle principali rivoluzioni nel mondo dello sviluppo software, soprattutto per quanto riguarda l'ambiente che comprende i sistemi operativi di Microsoft, e cioè delle varie versioni di Windows, che oggi abbracciano anche il mondo mobile.
Ma C# e .NET non sono assolutamente legati e limitati al mondo, un tempo chiuso, di **Windows**: l'iniziativa **.NET Core** rappresenta oggi la visione multipiattaforma (supportando oltre al sistema operativo suddetto anche **MacOS** e **Linux**) e open source di

Microsoft, che ora abbraccia quindi lo sviluppo server, cloud, web, passando naturalmente per il mondo delle applicazioni per il desktop, fino ad arrivare al lato mobile con **Xamarin**, l'ambiente di esecuzione dedicato alle applicazioni per dispositivi **Android** e **iOS**.
Detto ciò, C# può quindi essere considerato a tutti gli effetti un linguaggio di programmazione cross-platform.
Se vi state chiedendo se vale la pena sviluppare per .NET, riflettete sulle seguenti statistiche: più del 90% dei personal computer nel mondo hanno una versione di .NET installata, quindi si parla di oltre un miliardo di potenziali utenti. Su ognuna di queste macchine è possibile eseguire un'applicazione scritta in C#.
Il linguaggio C# è, fra quelli che è possibile utilizzare per lo sviluppo .NET, quello che riflette maggiormente le caratteristiche peculiari della piattaforma, in quanto nato con essa e per essa, e ne è quindi riconosciuto come il linguaggio principe. A oggi esso è giunto alla versione denominata **C# 7**.
Il **Framework .NET**, sviluppato praticamente di pari passo, segue una numerazione differente, e la sua versione più recente, al momento della stampa, è la **4.7**.
Non possono naturalmente mancare i riferimenti alle varie implementazioni dell'ambiente .NET, e quindi i confronti per capire le differenze e le peculiarità di **.NET Standard**, che è l'insieme di API che tutte le piattaforme di esecuzione .NET devono implementare, e .NET Core.
Il libro che state iniziando a leggere esporrà quindi le caratteristiche del linguaggio C#, aggiornate all'ultima versione disponibile al momento della sua stesura, che è come detto la 7, e di .NET Framework 4.7, utilizzando come ambiente di sviluppo la versione **Visual Studio 2017**, non tralasciando strumenti per altri sistemi, come **Visual Studio Code**. Anzi, se ne avete la possibilità, vi consiglio di provare a utilizzare quest'ultimo sui vari ambienti ove esso è disponibile, e cioè Linux, MacOS, e naturalmente lo stesso Windows.

A chi si rivolge il libro

Questo libro intende rivolgersi sia al lettore che si avvicina per la prima volta al mondo della programmazione, sia a quello che invece possiede già una qualsivoglia esperienza in tale ambito, magari con linguaggi differenti da C# e su piattaforme diverse da .NET. Alcuni concetti infatti sono comuni al mondo della programmazione orientata agli oggetti e quindi i capitoli iniziali che espongono tale paradigma di sviluppo possono essere letti in maniera rapida per arrivare al cuore della programmazione .NET in C#.
Scopo del libro è comunque quello di affrontare con la maggior precisione e approfondimento possibile i concetti trattati, e costituire quindi un riferimento completo del linguaggio C#, anche per il programmatore già esperto, che vuole avere a portata di

pagina una guida rapida, a cui fare riferimento per chiarire dubbi, per trovare risposte a domande e questioni più avanzate.
Inoltre, per chi, come il sottoscritto, vive e lavora nel mondo della programmazione da decenni, e quindi affronta l'argomento con ancora maggiore passione, ho cercato di trovare e fornire degli spunti e delle curiosità legate a C# e .NET lungo il diramarsi delle pagine e dei capitoli, facendo notare qual è stata l'evoluzione del linguaggio lungo le sue varie versioni, e indicando appunto in quale di esse ogni nuova funzionalità o caratteristica è stata introdotta.

Struttura del libro

Il libro è strutturato in maniera da permettere anche a chi non ha mai programmato di iniziare tale attività in maniera proficua, iniziando quindi dalle basi del linguaggio C#, fino ad arrivare ai concetti più complessi, e coprendo ogni novità introdotta dalle varie versioni fino ad arrivare alla più recente, denominata C# 7, permettendo di padroneggiare così ogni argomento che riguardi la programmazione .NET, anche quelli non esplicitamente trattati in questo testo.
Ogni nuova funzionalità introdotta da C# 7 è evidenziata e riportata anche nell'indice analitico, in maniera che anche coloro che fossero già in possesso delle edizioni passate del libro possano avere un rapido e completo riferimento di tutte le novità del linguaggio.
La prima parte del libro, costituita dai primi cinque capitoli, introdurrà la piattaforma .NET e le caratteristiche del linguaggio C# puro, iniziando quindi dalle parole chiave, dalla sintassi con cui si scrivono i programmi, introducendo i tipi fondamentali .NET, le espressioni e gli operatori, e i costrutti per controllare il flusso di esecuzione dei programmi.
Il **capitolo 1** esegue una prima panoramica di .NET e della sua architettura, mostrando anche i concetti di base della compilazione ed esecuzione dei programmi scritti in C#, ed elencando poi gli strumenti di programmazione che saranno utilizzati nel resto del libro e in ogni attività di sviluppo fatta come sviluppatori C#. In questa edizione oltre a Visual Studio e Windows, data la nuova natura multipiattaforma, si parla anche di .NET Core e Visual Studio Code, e quindi di Linux e MacOS.
Il **capitolo 2** mostra un primo programma C# e ne analizza il funzionamento dopo averlo compilato mediante strumenti come il compilatore a riga di comando e gli ambienti di sviluppo Visual Studio 2017 e Visual Studio Code. Lo stesso capitolo introduce la sintassi di base del linguaggio e i suoi elementi.
Il **capitolo 3** espone il sistema di tipi di .NET e le varie categorie di tali tipi creabili e utilizzabili in C#.

Nel **capitolo 4** si vedrà come scrivere espressioni più o meno complesse all'interno di un programma, utilizzando i vari operatori messi a disposizione dal linguaggio.
Il **capitolo 5** invece mostrerà come controllare l'esecuzione di un programma, utilizzando gli appositi costrutti e istruzioni di controllo del flusso.
A partire dal **capitolo 6** si entra nel mondo della programmazione a oggetti in C#, e quindi esso introduce concetti che permettono l'implementazione di classi personalizzate e struct, e quindi dei vari membri che ne possono costituire la struttura.
Il **capitolo 7** è la logica continuazione del precedente e approfondisce altri concetti della programmazione a oggetti, in particolare quelli di ereditarietà e polimorfismo, e presenta quello di interfaccia, il tutto allo scopo di realizzare complesse gerarchie di classi.
Si passa poi a concetti sempre più avanzati e nel **capitolo 8** viene introdotta la gestione delle cosiddette eccezioni, cioè delle situazioni di errore che si possono verificare durante l'esecuzione dei programmi.
Il **capitolo 9** tratta le collezioni di oggetti e la gestione di tipi parametrici mediante il meccanismo dei cosiddetti tipi generici.
Il **capitolo 10** tratta un argomento fondamentale di C#, come quello della programmazione a eventi e dei metodi di gestione degli stessi, e argomenti strettamente legati a questi, come i delegate, i metodi anonimi e le espressioni lambda.
Avanzando lungo i capitoli si copriranno tutte le sfaccettature del linguaggio C#, introdotte nelle sue varie versioni.
Quindi il **capitolo 11** include anche argomenti come LINQ, che permette l'interrogazione di varie forme di dati mediante una nuova sintassi e nuovi metodi e tipi, introdotti in C# 3.0.
Il **capitolo 12** pone l'accento sulle prestazioni e affronta argomenti come il multithreading, la programmazione parallela e quella asincrona, per sfruttare i moderni processori dotati di più core.
Fra i formati di dati più utilizzati nelle applicazioni vi è senz'altro l'XML. Il **capitolo 13** esplora le funzioni utilizzabili da C# per manipolare tale formato, partendo dal classico XML DOM, passando per XPath fino a LINQ to XML.
Il **capitolo 14** scende in profondità nei meandri della composizione dei tipi. Infatti il cosiddetto meccanismo di Reflection permette di analizzare ogni aspetto di un oggetto, a tempo di esecuzione. Nello stesso capitolo si vedranno anche gli attributi e il loro utilizzo.
Il **capitolo 15** è uno dei più lunghi, in quanto affronta un argomento importante in ambito pratico come l'accesso ai dati, esplorando l'input/output su file, e le tecnologie ADO.NET, LINQ to SQL e Entity Framework per l'accesso ai database relazionali.
Nel **capitolo 16**, viene affrontato uno degli argomenti chiave delle ultime versioni di C#, cioè la .NET Compiler Platform, o Roslyn, mostrando come utilizzare i servizi messi a disposizione del compilatore nelle applicazioni, o per scrivere estensioni di Visual Studio.

Nell'ultimo, il **Capitolo 17**, tramite lo sviluppo di qualche applicazione pratica in vari ambiti e sistemi operativi, viene mostrata infine la versatilità di C# e .NET.
L'**appendice A** è dedicata all'utilizzo delle classi necessarie per lavorare con stringhe e testi e alle espresioni regolari, in quanto sono un argomento che trova parecchia utilità nella pratica di tutti i giorni.
L'**appendice B** mostra dei rapidi cenni sulla programmazione con i puntatori, quindi in cosiddetto contesto unsafe, e per l'utilizzo da codice gestito di funzioni native, per mezzo dei servizi P/Invoke.
L'**appendice C** contiene le risposte alle domande di riepilogo poste alla fine di ogni capitolo.
In tal modo si sarà compiuto un completo viaggio all'interno di tutte le caratteristiche e potenzialità offerte dal linguaggio C# e dalla piattaforma .NET, senza naturalmente la pretesa di essere totalmente esaustivi, dati i limiti imposti dalla lunghezza del testo.

Le novità di C# 7

Il libro è aggiornato a tutte le nuove funzionalità introdotte con la versione 7 (o 7.0) del linguaggio C#. Per chi volesse un rapido riferimento di tali novità ecco un elenco esaustivo:

- valori letterali per numeri binari;
- separatori di cifre;
- pattern matching con operatore is e con istruzione switch;
- dichiarazione inline di variabili out;
- scarto (discard) dei parametri out;
- variabili locali e di uscita per riferimento;
- funzioni locali;
- corpo di espressione in nuovi tipi di membri di classe;
- espressioni throw;
- tuple: tipo nativo e sintassi;
- decostruzione tuple;
- metodi di decostruzione per i tipi;
- tipi di ritorno personalizzati per metodi async.

L'indice analitico alla fine del libro, alla voce C# 7, vi permetterà di trovare facilmente le pagine in cui ognuno di questi argomenti viene trattato.

Esempi pratici e capitoli bonus

Ogni capitolo del libro contiene esempi pratici che potete scrivere e compilare autonomamente, sia utilizzando i compilatori a riga di comando, sia all'interno di ambienti

di sviluppo come Visual Studio Code e Visual Studio; tutte le versioni di quest'ultimo, e in particolare la **2017**, sono adeguate se non avete intenzione di utilizzare le caratteristiche del linguaggio introdotte con C# 7 (ma se state leggendo questo libro probabilmente vorrete farlo!).
Per chi non fosse provvisto di una licenza professionale di Visual Studio 2017 non c'è alcun problema: come avremo modo di vedere già dai primi capitoli, è infatti disponibile una versione denominata **Community**, liberamente scaricabile da internet e che permette di sviluppare e distribuire applicazioni scritte in C#, per ogni ambiente, per esempio desktop, web, o mobile.
Le indicazioni e i link da cui scaricare gli esempi completi sono inoltre disponibili sul mio sito internet, alla pagina http://www.antoniopelleriti.it/page/libro-csharp, dotati di file di soluzione .sln, che potete quindi aprire direttamente in Visual Studio.
Sulla stessa pagina, troverete anche capitoli bonus che, per limiti di spazio o per aggiornare il testo con argomenti usciti dopo la stampa, non fanno parte della versione finale del libro.

Errata corrige

Sebbene il libro sia stato letto e riletto, qualche errore può sempre sfuggire! Quindi eventuali correzioni potete trovarle sempre sul sito dedicato http://www.antoniopelleriti.it/page/libro-csharp.
Sullo stesso sito e sulla pagina facebook a esso dedicata, https://www.facebook.com/programmare.con.csharp, potete effettuare le vostre segnalazioni di errori e imprecisioni, e proporre magari suggerimenti per le prossime, spero numerose, edizioni del libro!

L'autore

Antonio Pelleriti è ingegnere informatico e si occupa da diversi anni di sviluppo software, su varie piattaforme.
In particolare il .NET Framework e le tecnologie a esso correlate sono i suoi principali interessi fin dal rilascio della prima beta della piattaforma Microsoft, quindi da oltre 15 anni. In tale ambito il riconoscimento più importante è la nomina a **Microsoft MVP** per .NET da gennaio 2015, che oggi è diventata una nomina a Microsoft MVP per Visual Studio (il link al profilo pubblico è https://mvp.microsoft.com/it-it/PublicProfile/5001181?fullName=Antonio%20Pelleriti).
Autore di numerose pubblicazioni per riviste di programmazione e di guide tascabili dedicate al mondo .NET, per Edizioni FAG ha già pubblicato nel 2011 il libro "Silverlight 4, guida alla programmazione", e per LSWR, la prima edizione di "Programmare con C# 5, guida completa" (2014) e la seconda "Programmare con C# 6, guida completa" (2016).

Dopo aver girato in lungo e in largo la penisola, partendo da Braidi, suo amato e ridente paesello abbarbicato sui monti Nebrodi, e lavorando per primarie aziende nazionali e multinazionali, ritorna in Sicilia, naturalmente continuando a seguire ogni aspetto di sviluppo legato alla piattaforma .NET, e lavorando a progetti software di ogni dimensione e tipo.
Il suo sito personale su cui pubblica articoli e pillole di programmazione legate sempre a .NET e C# è www.antoniopelleriti.it.

Ringraziamenti

Scrivere un libro sul linguaggio che si studia e utilizza per lavoro fin dalla sua apparizione nel mondo dello sviluppo software può essere considerato un sogno che si realizza.
Il successo della prima edizione è dunque una soddisfazione enorme, soprattutto per i feedback ricevuti dai lettori con le loro recensioni e tramite i loro messaggi ed email. In particolare un grazie enorme a tutti coloro che mi hanno fatto scovare errori e imprecisioni e che mi hanno donato i loro consigli per migliorare il testo in ogni suo aspetto. Il fatto che dopo poco più di un anno mi sono ritrovato a scrivere l'edizione aggiornata, è merito di tutti loro.
Non posso che ringraziare poi ancora una volta **Marco Aleotti** e tutto il team di **Edizioni LSWR**, per la rinnovata fiducia e per la collaborazione.
Ringrazio la mia gatta **Kiya**, per la compagnia nelle varie sessioni di scrittura notturne o mattiniere (se trovate qualche refuso potrebbe essere stato causato da una delle sue incursioni sulla mia tastiera).
E naturalmente, come sempre in maniera anticipata, ringrazio te, **lettore**, che stai tenendo in mano questo libro cartaceo, o lo stai sfogliando virtualmente su un PC o altro dispositivo, e che quindi hai già riposto fiducia nei miei confronti (magari per la terza volta, se hai già letto le prime due edizioni!).

E come tutte le cose che faccio: a **Caterina** e **Matilda**.

```
while(true)
{
  ViAdoro();
}
```

1

C# e la piattaforma .NET

.NET è una piattaforma di sviluppo ed esecuzione di applicazioni disponibile su diversi sistemi operativi, e **C#** è il suo linguaggio di programmazione principe, **multiparadigma** e principalmente **orientato agli oggetti**.

La piattaforma .NET nasce alla fine degli anni '90, quando Microsoft inizia a lavorare a un nuovo e rivoluzionario modello di sviluppo e programmazione dei propri sistemi operativi, più semplice e al contempo più potente rispetto a quello fino ad allora utilizzato dai programmatori del mondo Windows.
A quei tempi il nome con cui Microsoft si riferiva a tale nuova piattaforma, ancora in fase di sviluppo, era *NGWS*, acronimo di *Next Generation Windows Services,* che stava proprio a indicare le intenzioni di creare una nuova generazione di strumenti con cui si sarebbero sviluppati software e servizi per il sistema operativo Windows.
Assieme al nuovo framework, che assume il nome definitivo di **.NET Framework** quando viene ufficialmente annunciato al grande pubblico, viene progettato un nuovo linguaggio di programmazione, anch'esso prima denominato con il nome provvisorio **COOL** (*C-Like Object Oriented Language*), per poi essere battezzato come l'ormai noto **C#** (C Sharp) e che nasce per diventare il linguaggio principe per lo sviluppo di software sulla nuova piattaforma.
Sebbene il nome .NET possa essere fuorviante per chi si stia approcciando per la prima volta al framework, in quanto potrebbe richiamare concetti legati esclusivamente al mondo internet, esso deriva dalla volontà, dalla convinzione e quindi dal conseguente impegno che Microsoft intende profondere nel mondo delle applicazioni sempre più di-

stribuite e interconnesse, siano esse appartenenti al mondo **desktop**, ma anche al **web**, al **cloud**, al variegato universo dei dispositivi **mobili** come smartphone e tablet, all'**Internet of Things**, e oggi anche a quello dei diversi sistemi operativi, non solo Windows. Essendo quindi C# così fortemente e quasi indissolubilmente legato alla piattaforma .NET, usandola come sua destinazione, è necessario che chiunque intenda iniziare a sviluppare in tale linguaggio, o che aspiri comunque a passare a un livello di sviluppo avanzato, comprenda che la conoscenza dei vari aspetti di .NET costituisce un prerequisito fondamentale della programmazione in C#.

Prima di .NET

Perché a un certo punto della storia dello sviluppo software è sorta la necessità di inventarsi una nuova piattaforma di sviluppo e di esecuzione delle applicazioni e un nuovo linguaggio di programmazione?
Come in altri ambiti, la storia stessa può aiutarci a fornire la risposta.
Prima dell'avvento di .NET, il mondo degli sviluppatori Windows doveva barcamenarsi all'interno di una giungla di diverse tecnologie, tecniche e linguaggi.
Si poteva scegliere di programmare servendosi delle API *Win32*, che costituiscono l'interfaccia di programmazione del sistema operativo in linguaggio C, oppure fare uso delle Microsoft Foundation Classes (*MFC*) in C++, altri ancora sceglievano *COM*, che era la modalità di sviluppo indipendente dal linguaggio scelto per creare oggetti e componenti e che forse può essere considerato il predecessore di .NET, almeno negli intenti e negli scopi.
Dalla prima versione di Windows, la 1.0. rilasciata nel 1985, attraverso le varie versioni susseguitesi nel corso degli anni, per semplificare lo sviluppo di applicazioni e quindi aumentare la produttività (si pensi che un Hello World che mostri una finestra nella prima versione di Windows era lungo all'incirca 150 linee di codice), Microsoft introduce diverse novità: nuove versioni di API a 32 bit, nuove funzioni, librerie, strumenti, tecnologie e anche la possibilità di programmare utilizzando linguaggi diversi dal C, al fine di nascondere (o almeno per cercare di farlo!) le complessità di più basso livello.
Vedono così la luce, in ordine sparso e senza pretese di essere esaustivi, sigle e nomi come Win32, COM, ATL, DDE, ActiveX, MFC, Visual Basic ecc.
Arrivati alla fine degli anni '90, le necessità e i possibili ambiti applicativi su cui gli sviluppatori possono e devono cimentarsi sono diventati parecchi, e si apre inoltre un nuovo orizzonte su cui diventa necessario affacciarsi in maniera sempre più frequente, il web.

L'avvento di .NET

Quando in Microsoft si inizia a progettare .NET, l'obiettivo da raggiungere è ben chiaro: .NET vuole essere innanzitutto un ambiente unificato che supporti lo sviluppo e l'esecuzione di applicazioni.
Esso è pensato fin dall'inizio come soluzione multipiattaforma, non legata ad alcun sistema operativo e ad alcun microprocessore in particolare, anche se Microsoft ha sempre rilasciato l'ambiente di esecuzione su propri sistemi Windows. Oggi, la visione strategica più recente e aggiornata di Microsoft stessa, ha dato vita a una versione veramente cross-platform e open source della piattaforma .NET, denominata .NET Core, e basata su precisi **standard .NET**, che specificano i concetti chiave, e che consentirà quindi di essere eseguita anche su sistemi Linux e MacOS o su quelli che implementeranno gli standard stessi.

NOTA

Un'altra implementazione non Microsoft, molto nota e anch'essa open source, è quella denominata **Mono** (vedere http://www.mono-project.com/ per approfondire), che fornisce una versione di .NET che, oltre a Windows, gira anche su sistemi Linux e MacOS, e che comprende anche il compilatore C# e strumenti di sviluppo come *MonoDevelop*.

Inoltre, ulteriori implementazioni consentono di sviluppare in C# applicazioni per le piattaforme mobile iOS e Android (rispettivamente chiamate *Xamarin.iOS* e *Xamarin.Android*, anch'esse nate dal suddetto Mono, infatti le prime versioni erano note con i nomi di MonoTouch e MonoAndroid). Allo stesso modo *Xamarin.Mac* permette di sviluppare applicazioni native per MacOS in C# e .NET.
Oltre a non essere limitato ad alcun sistema operativo, .NET non ha limiti di sorta nemmeno per il tipo di applicazioni che è possibile sviluppare; infatti consente di programmare ed eseguire applicazioni per il mondo desktop, per il web, per il mobile e, in generale, per ogni ambiente su cui sarà possibile implementare l'ambiente di esecuzione .NET. Infine, come già detto, è anche possibile scegliere il proprio linguaggio preferito per lo sviluppo, fra quelli naturalmente che consentono una compilazione per l'ambiente .NET (ve ne sono a decine oltre a C#). Una delle potenzialità più rilevanti di tale caratteristica multilinguaggio è che sarà possibile utilizzare, da un'applicazione scritta in C#, anche librerie o componenti scritti in altri linguaggi.

NOTA

Un elenco dei linguaggi di programmazione utilizzabili per lo sviluppo in .NET è disponibile al seguente link: https://en.wikipedia.org/wiki/List_of_CLI_languages.

Panoramica di .NET

.NET è una piattaforma generale per lo sviluppo di qualsiasi tipo di software, che comprende una libreria di oggetti di base riutilizzabili, e un ambiente di esecuzione che al suo interno gestisce l'intero ciclo di vita delle applicazioni, e fornisce loro i vari servizi necessari.

Essa integra diverse funzionalità fondamentali per gli sviluppatori, che permettono di aumentare la produttività, la qualità e la sicurezza delle proprie applicazioni. Inoltre fornisce strumenti pratici di alto livello (intendendo quello vicino allo sviluppatore stesso), ma al tempo stesso consente di interagire a basso livello alle risorse del sistema, come la memoria o le API native.

Secondo quelli che erano gli obiettivi progettuali del team .NET, il framework possiede le seguenti caratteristiche di base:

- fornire una **piattaforma di sviluppo moderna e orientata agli oggetti**, ma con diverse opzioni per il paradigma di programmazione **funzionale**;
- supporto e **interoperabilità** fra diversi **linguaggi** di programmazione: i compilatori di tutti i linguaggi utilizzabili in .NET (C#, F# e Visual Basic ad esempio) generano un codice intermedio, denominato Common Intermediate Language (CIL), che, a sua volta, viene compilato in fase di esecuzione dall'ambiente di esecuzione Common Language Runtime. Con questa funzionalità, le funzioni scritte in un linguaggio sono accessibili ad altri linguaggi e i programmatori possono concentrarsi sulla creazione di applicazioni nei propri linguaggi preferiti;
- **multipiattaforma**: il formato binario e le istruzioni sono indipendenti dal sistema operativo e dal processore, è possibile così creare delle applicazioni e librerie portabili che funzionano su diverse piattaforme come Windows, Windows Phone, Xbox, Android, iOS e così via;
- un **sistema di tipi comune**: nei linguaggi tradizionali i tipi supportati sono definiti dal compilatore, il che rende difficile l'interoperabilità fra linguaggi diversi. .NET definisce un sistema di tipi (Common Type System) che sono accessibili e utilizzabili da qualunque linguaggio che supporti il .NET framework;
- un'estesa **libreria di classi di base**: i programmatori hanno a disposizione e possono utilizzare una libreria di tipi e relativi membri facilmente accessibili, anziché dover scrivere grandi quantità di codice per gestire operazioni comuni di programmazione di basso livello. Inoltre ogni implementazione include librerie o sottosistemi specifici per particolari aree applicative: per esempio in .NET Framework si troveranno ASP.NET per il web, ADO.NET o Entity Framework per l'accesso a database, WCF per lo sviluppo di applicazioni orientate ai servizi, e così via;

- miglioramento e semplificazione della fase di **distribuzione** e **versionamento** delle applicazioni, con gestione della compatibilità di versione: per esempio una applicazione sviluppata per una data versione di .NET Framework continuerà a funzionare su tutte le versioni successive;
- un **ambiente di esecuzione** dalle performance elevate e ottimizzate per ogni piattaforma su cui esso è supportato;
- esecuzione **side-by-side**: è possibile la coesistenza di diverse versioni dell'ambiente di esecuzione di .NET sullo stesso computer, in maniera che ogni applicazione o ogni particolare versione di un'applicazione, possa essere eseguita nell'ambiente per il quale era stata sviluppata;
- **gestione semplificata della memoria**, grazie all'esecuzione automatica di un garbage collector: il programmatore è esentato dal doversi preoccupare di operazioni come allocare e rilasciare la memoria manualmente, è l'ambiente di esecuzione che si occuperà di tutto.

Fra i linguaggi supportati, C# nasce assieme a .NET e riflette dunque ognuna delle caratteristiche progettuali del framework.
Ogni programma scritto in C# è eseguibile solo ed esclusivamente all'interno di un ambiente di esecuzione .NET. Per tale motivo una comprensione approfondita di .NET e dei suoi singoli componenti (esposti nei prossimi paragrafi) è fondamentale per iniziare a programmare in C# e se si vuole riuscire a ottenere un livello elevato di conoscenza e messa in pratica efficace delle sue caratteristiche.

Architettura di .NET

.NET è costituito da un insieme di componenti principali. Innanzitutto è presente una libreria di base, denominata .NET Standard Library, che comprende un ampio e variegato insieme di classi e API. Tale libreria è implementata da Microsoft con tre ambienti di esecuzione o runtime .NET: .NET Framework, .NET Core e Mono per Xamarin.
Ogni implementazione .NET è poi basata su una infrastruttura comune, che comprende vari strumenti e componenti, che includono fra l'altro i linguaggi e relativi compilatori, componenti del runtime come il garbage collector e il compilatore JIT, strumenti per la compilazione e gestione dei progetti, il sistema di gestione dei pacchetti NuGet e così via.
La Figura 1.1 mostra graficamente come i componenti suddetti sono collocati nell'architettura appena esposta.

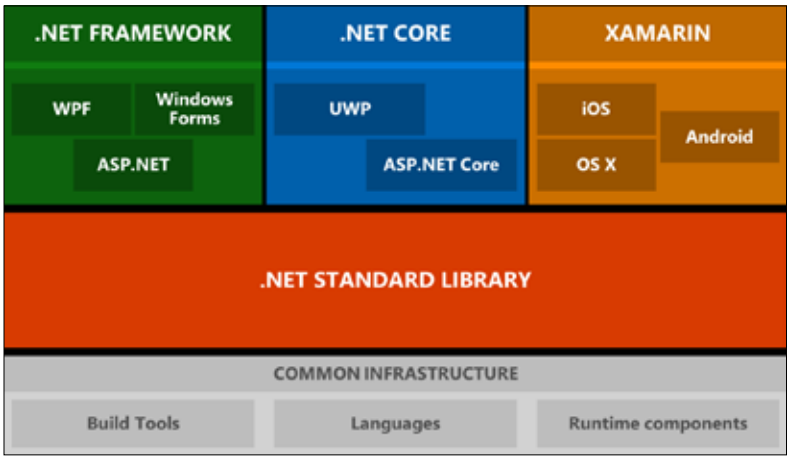

Figura 1.1 – Architettura dei componenti di .NET.

Nel seguito saranno spiegati in maggior dettaglio i componenti appena visti.

.NET Runtime

Microsoft ha sviluppato e mantiene tre ambienti di esecuzione o runtime: .NET Framework, .NET Core, e Mono per Xamarin.

Il primo, **.NET Framework**, è il runtime originale, il primo a essere sviluppato a partire dal 2002, ed è quindi quello che in generale è stato utilizzato da ogni sviluppatore .NET, e continuerà a rappresentare la scelta naturale per la maggior parte degli scenari esistenti. Esso contiene in particolare classi e API specifiche dell'ambito Windows, e quindi fornisce librerie ed è ottimizzato per lo sviluppo di applicazioni desktop Windows Forms e WPF, ma anche per lo sviluppo web in ASP.NET. Nei prossimi paragrafi del capitolo viene dedicato maggior spazio a tale ambiente, dando un'occhiata a esso e ai suoi componenti in maggior dettaglio.

.NET Core è l'ambiente di esecuzione open source e può essere considerato la versione multipiattaforma di .NET Framework, particolarmente ottimizzato per servizi e applicazioni lato server, ed è anche il runtime utilizzato da ASP.NET Core, dalla piattaforma Universal Windows Platform e da Xamarin.Forms. Esso è supportato su Windows, MacOS e Linux.

L'ambiente di esecuzione vero e proprio di .NET Core (inteso come piattaforma) è detto *CoreCLR*, mentre le librerie di base della piattaforma sono racchiuse nel termine *CoreFX*. È possibile utilizzare i linguaggi C# e F# (a breve Visual Basic) per scrivere codice per .NET Core.

NOTA

I componenti di .NET Core (in particolare CoreCLR e CoreFX) sono open source su GitHub: le pagine ufficiali e il codice sorgente sono raggiungibili agli indirizzi https://github.com/dotnet/coreclr e https://github.com/dotnet/corefx rispettivamente.

Una delle principali caratteristiche di .NET Core è la sua modularità, tanto che esso è completamente distribuibile con le applicazioni, per mezzo dei relativi pacchetti NuGet, senza quindi la necessità di dover eseguire in un ambiente già dotato di una qualunque versione di un .NET Runtime. In tal modo inoltre vengono ottimizzati la memoria e lo spazio disco altrimenti necessario per distribuire e installare un intero insieme di librerie, limitandosi a quelle che effettivamente sono utilizzate dall'applicazione in oggetto.
Mono è il runtime multipiattaforma utilizzato e ottimizzato per lo sviluppo di app Xamarin, cioè applicazioni destinate a essere eseguite su sistemi iOS, Android e Mac.
Originariamente Mono era la versione multipiattaforma (destinata in particolare a Linux) del .NET Framework sviluppato originariamente da Ximian e Novell, e basato sulle sue specifiche standard (in particolare ECMA 334 e 335) pubblicate da Microsoft per chi volesse implementare una versione perfettamente compatibile.

.NET Standard Library

La libreria .NET Standard è costituita da un insieme di API, implementate da un runtime .NET. Mediante la specifica dei contratti che tali API devono rispettare, ogni runtime che implementa la libreria .NET Standard può eseguire lo stesso identico codice.
La libreria .NET Standard è inoltre una destinazione di compilazione, ed è in questo caso chiamata solo .NET Standard: così .NET Standard è una specifica formale delle API che sono disponibili per un dato runtime.
Se un runtime .NET supporta una specifica versione .NET Standard, significa che ogni versione precedente è altresì supportata.
Per esempio la versione di .NET Framework che al momento della stesura del libro supporta la versione 2.0 di .NET Standard è il NET Framework 4.6.1. Ciò significa che esso supporta anche tutte le versioni precedenti da 1.0 a 1.6.
In generale quindi, nello scegliere la versione .NET Standard da utilizzare per le proprie applicazioni, bisogna accettare il seguente compromesso: più è alta la versione maggiori sono le API a disposizione, più è bassa maggiori sono le piattaforme che la supportano e che quindi potranno eseguire l'applicazione.

NOTA

Per fornire un riferimento unico agli sviluppatori che ricercano documentazione sulle API supportate dai vari ambienti, Microsoft ha messo a disposizione uno strumento chiamato *.NET API Browser*, che permette di navigare facilmente fra le varie classi e consente eventualmente di filtrarle specificando il runtime supportato. Tale strumento è raggiungibile all'URL https://docs.microsoft.com/dotnet/api.

Il .NET Framework

Il .NET Framework è la piattaforma completa e continua a essere la scelta naturale per creare applicazioni e librerie per i sistemi Windows desktop, in particolare quelle dotate di una classica interfaccia grafica a finestre, ma essa fornisce tutto il necessario per lo sviluppo e l'esecuzione di applicazioni eterogenee: oltre al desktop, anche mobile, web, servizi, cloud e così via.

Il .NET Framework consiste di due componenti principali, il suo **ambiente di esecuzione** e una estesa **libreria di classi**, denominata *Framework Class Library*. Il nucleo di questa libreria è un insieme di altre classi di base, detto per questo *Base Class Library* (BCL).

La Figura 1.2 mostra una visione di insieme dell'architettura a livelli di .NET Framework.

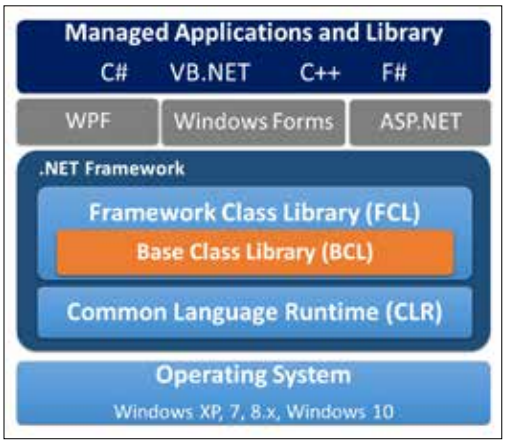

Figura 1.2 – Architettura a livelli del .NET Framework.

Al di sotto del CLR vi è il sistema operativo, che può essere uno qualunque dei sistemi per i quali esiste un'implementazione del CLR stesso. In particolare il runtime di .NET Framework è incluso in tutte le recenti versioni di Windows e consente quindi di eseguire applicazioni .NET Framework senza la necessità di installazioni aggiuntive.

Il .NET Framework a sua volta è utilizzato da ogni applicazione o tecnologia che sfrutta la sua libreria di classi. In generale infatti le applicazioni sfruttano diverse tecnologie, costituite da librerie di livello ancor più alto che implementano particolari funzionalità utili in un determinato ambito di sviluppo. Per esempio WPF e Windows Forms forniscono le classi per scrivere applicazioni per il desktop dotate di interfaccia grafica, mentre ASP.NET è la tecnologia per lo sviluppo web.
Per la scrittura del codice sorgente delle applicazioni può essere utilizzato uno qualunque dei linguaggi cosiddetti *.NET Enabled* (nella Figura 1.2 sono elencati solo i linguaggi supportati nativamente da Visual Studio 2017).
La discussione fatta nei prossimi paragrafi per il .NET Framework e i rispettivi componenti è in buona parte valida anche per .NET Core, Mono per Xamarin, o una qualunque implementazione .NET, in quanto i concetti di librerie di base e ambiente di esecuzione sono perfettamente analoghi.

Common Language Runtime

Il primo componente fondamentale di .NET è il suo *ambiente di esecuzione*, il cosiddetto **Common Language Runtime** (CLR) che costituisce una sorta di macchina virtuale all'interno della quale i programmi scritti in C# o in uno dei linguaggi supportati dalla piattaforma vengono eseguiti.
Il codice eseguito dal CLR viene detto *managed code*, o *codice gestito* (dal CLR appunto). Al contrario, il codice che non passa dal CLR viene detto *unmanaged code*, cioè codice non gestito, con il quale si intende dunque tutto il codice macchina, per esempio il codice nativo scritto sfruttando le API Win32 di Windows.

Compilazione ed esecuzione

Un programma scritto in C# prima di poter essere eseguito deve essere convertito in un linguaggio intermedio, chiamato **CIL** (*Common Intermediate Language*) o **IL** (*Intermediate Language*), indipendente dalla CPU e dal linguaggio (anche Visual Basic verrà convertito nello stesso IL), che è comprensibile dal CLR.
L'Intermediate Language è una sorta di linguaggio macchina, ma di livello molto più alto, in quanto possiede anche caratteristiche di linguaggio orientato agli oggetti, funzioni per l'utilizzo di array e per la gestione degli errori.
Il CLR non ha idea di quale linguaggio sia stato utilizzato per produrre il codice intermedio IL, e non gli interessa nemmeno. Il codice IL però non è ancora eseguibile direttamente dal sistema operativo.
Quindi, in un secondo processo di compilazione, il programma, ora in codice IL, viene trasformato nel linguaggio macchina specifico della piattaforma su cui esegue il CLR stesso. Tale processo viene chiamato *compilazione JIT* (*just in time*), ed è eseguita dal

JITCompiler o *Jitter*. Il Jitter si occupa di produrre codice specifico della piattaforma, per esempio se esso è in esecuzione su una versione x86 di Windows allora produrrà istruzioni x86.

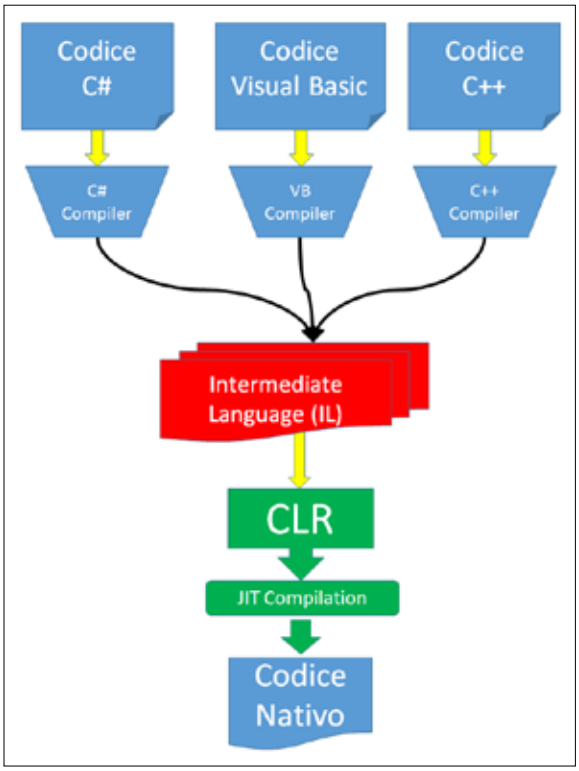

Figura 1.3 – Compilazione ed esecuzione in .NET.

A questo punto il sistema operativo sarà in grado di eseguire l'applicazione.
La modalità di compilazione *just in time* permette di ottenere performance superiori rispetto alla modalità di esecuzione di un linguaggio interpretato.

NOTA

Un'altra possibilità, sfruttata da molte applicazioni .NET, è quella di compilare utilizzando il compilatore .NET Native, generando direttamente codice nativo al posto di IL, migliorando le performance e lo sfruttamento della memoria, ma al costo di perdere la portabilità dell'applicazione.

Il compito del CLR però non termina qui: esso gestirà l'esecuzione delle applicazioni in una sorta di macchina virtuale, occupandosi di funzioni fondamentali come la ge-

stione della memoria, la gestione di eventuali errori durante l'esecuzione, la sicurezza e l'interoperabilità.
Sarebbe possibile anche scrivere un programma direttamente in codice IL e poi darlo in pasto al JIT Compiler, ma linguaggi di alto livello come C# hanno proprio il compito di rendere più semplice lo sviluppo.
I vantaggi del codice IL sono l'interoperabilità fra i linguaggi .NET e la possibilità di essere eseguito su diverse piattaforme, in quanto basterà avere un'implementazione specifica del CLR che converta lo stesso IL (uguale per tutte le piattaforme) nel codice macchina nativo della piattaforma su cui è in esecuzione.
Microsoft fornisce con la sua implementazione della piattaforma .NET anche un assembler di linguaggio IL chiamato **ILAsm** e il corrispondente disassemblatore, **ILDasm**, che permette di ottenere il codice IL a partire da un programma eseguibile.

Assembly

Quando un programma C# viene compilato, il codice IL ottenuto viene conservato all'interno di uno o più file detti *assembly*.
Un assembly è una unità logica che può essere direttamente eseguita dal sistema operativo (file .exe) oppure che può essere utilizzata da altri programmi come libreria di codice (file .dll).
La struttura logica è identica in entrambi i casi, l'unica differenza è che un assembly eseguibile contiene anche un punto di ingresso, il cosiddetto *entry-point*, che indica al CLR da dove iniziare l'esecuzione dell'applicazione.
Nonostante le estensioni utilizzate siano ancora .exe e .dll, il contenuto dei rispettivi file è differente dal formato nativo dei file eseguibili e delle librerie di Windows.
Ogni assembly infatti contiene oltre al codice intermedio anche un *manifest* che descrive l'assembly stesso, dei *metadati* che descrivono i tipi contenuti dell'assembly e delle eventuali *risorse* (ad esempio immagini, audio ecc.).
Grazie a tali metadati ogni assembly è completamente auto-descrittivo e non sono necessarie altre informazioni esterne per poterlo utilizzare. In parole povere sarà sufficiente copiare uno o più assembly che costituiscono un'applicazione su un computer per poterla eseguire (naturalmente sarà necessaria la presenza dell'ambiente di esecuzione del .NET Framework su tale computer).
La Figura 1.4 mostra il formato di un assembly .NET e del suo contenuto.
In realtà il contenuto di un assembly può anche essere suddiviso in diversi file, per esempio separando le risorse in un altro assembly, in maniera da poter eventualmente sostituire solo la parte con il codice IL, senza dover necessariamente ridistribuire un corposo insieme di altre risorse non modificate.