- Antonio Pelleriti -

PROGRAMMARECON

C# 6

Guida completa

2ª EDIZIONE

BESTSELLER

Compilatore e ambiente di sviluppo Visual Studio 2015 >>

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

Sviluppo per Windows, DB, LINQ, XML, Compiler API >>

Sintassi e costrutti del linguaggio >>

*pro DigitalLifeStyle

PROGRAMMARECON

C# 6

Guida completa

Antonio Pelleriti

EDIZIONI
LSWR

Programmare con C# 6 | Guida completa

Autore: Antonio Pelleriti

Collana: *pro DigitalLifeStyle

Editor in Chief: Marco Aleotti
Progetto grafico: Roberta Venturieri
Immagine di copertina: © djvstock | Thinkstock
Realizzazione editoriale e impaginazione: Studio Dedita di Davide Gianetti

ISBN: 978-88-6895-308-9

EDIZIONI LSWR

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

Printed in Italy

Finito di stampare nel mese di marzo 2016 presso "LegoDigit" Srl., Lavis (TN)

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

Sommario

Introduzione

Il linguaggio di programmazione **C#** è ormai un attore protagonista maturo ed esperto sul palcoscenico dello sviluppo software, non solo in ambito Microsoft. Esso è infatti sulla scena da quasi quindici anni e ha continuato a evolversi costantemente dal 2000 a oggi, introducendo, di volta in volta, in ognuna delle versioni rilasciate, caratteristiche e funzionalità nuove, volte a migliorare la produttività dello sviluppatore che lo utilizza nel proprio lavoro quotidiano.
In tale evoluzione il linguaggio non è mai divenuto pesante e complesso e, anzi, ha mantenuto la semplicità e la freschezza che gli hanno permesso, dividendo naturalmente il merito con la piattaforma **.NET Framework** per cui e con cui è nato, di poter affrontare e risolvere problemi legati ad ambiti di sviluppo appartenenti a mondi differenti, sia dal punto di vista della piattaforma di esecuzione sia da quello prettamente pratico e legato all'ambito applicativo.
Per tale motivo C# e .NET costituiscono oggi una delle principali scelte per chi vuole creare un software che giri sul desktop di un personal computer, come applicazione web all'interno di un browser internet o, ancora, come app installata su uno smartphone o su un tablet, spaziando in qualunque caso in degli **scenari applicativi** estremamente eterogenei: dal mondo dell'industria a quello dei software di produttività e gestione aziendale, passando per i videogame e i sistemi di commercio elettronico.
La piattaforma .NET ha costituito una delle principali rivoluzioni nel mondo dello sviluppo software, soprattutto per quanto riguarda l'ambiente costituito dai sistemi operativi di **Microsoft**, cioè delle varie versioni di **Windows**, che oggi abbracciano anche il mondo mobile.
Le ultime release, a partire da quelle della famiglia Windows 8.x e oggi Windows 10, permettono di sviluppare app che gli utenti possono acquistare e scaricare direttamente dal **Windows Store**, un negozio digitale analogo all'App Store di Apple o al Play Store di Google, e che quindi costituisce una vetrina che si affaccia su un mercato immenso di potenziali clienti.

Ma C# e .NET non sono assolutamente legati e limitati al mondo Windows e di Microsoft. I componenti fondamentali di .NET, oltre al linguaggio C#, sono divenuti degli standard open e quindi liberamente implementabili anche su sistemi differenti, opportunità che, come dimostra l'implementazione open source **Mono** che può girare anche su sistemi Linux o Mac, non è rimasta solo una fantasia o un'opportunità mai sfruttata.
Le recenti dichiarazioni di Microsoft hanno inoltre aperto strade un tempo raramente percorse, abbracciando il mondo **open source**. La nuova visione di Microsoft ha quindi dato vita a un nuovo filone di .NET, denominato nella sua interezza .NET 2015, all'interno del quale trova posto **.NET Core**, un framework open source, modulare, portabile e quindi dedicato principalmente allo sviluppo cross platform. In tal modo esso potrà facilmente essere portato su sistemi e architetture diverse, come **Linux** e **MacOS-X**.
Fra i componenti principali di .NET Core sono da citare i nuovi compilatori (fra cui quello di C#), riscritti completamente in .NET e resi anch'essi disponibili alla comunità di sviluppatori come progetti open source. Visual Studio 2015 stesso è basato su questa nuova piattaforma, chiamata **.NET Compiler Platform**, che a lungo è stata conosciuta come **Roslyn**, e che è stata messa anch'essa a disposizione degli sviluppatori per sfruttare le stesse API e librerie che costituiscono il cuore pulsante dell'ambiente di sviluppo di Microsoft.
Il parco mondiale di macchine dotate della piattaforma .NET supera il miliardo, secondo recenti conteggi fatti per mezzo di Microsoft Update, e quindi praticamente esatti. Su ognuna di queste macchine è possibile eseguire un'applicazione scritta in C#.
Volgendo lo sguardo verso un altro dei mercati che gli sviluppatori non possono più ignorare, quello degli smartphone e dei tablet, il sistema operativo **Windows Phone** ha riscosso un successo sempre crescente e l'Italia è una delle nazioni in cui tale successo è addirittura più marcato. Ancora meglio, con **Windows 10** il sistema operativo si è mosso verso un'unificazione sempre più spinta, consentendo l'esecuzione di applicazioni scritte una volta sola, su piattaforme hardware estremamente eterogenee e che, a ragion veduta, sono state denominate **applicazioni universali**.
Con C# potrete sviluppare applicazioni di questo genere e, se volete abbracciare un mercato ancora più ampio, potrete addirittura mantenere lo stesso linguaggio per portare le vostre app nei territori un tempo ostili di Apple e di Android, sfruttando strumenti come quelli forniti da **Xamarin**, basati su C#.
Il linguaggio C# è, fra quelli che è possibile utilizzare per lo sviluppo .NET, quello che riflette maggiormente le caratteristiche distintive 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# 6**.
Il **Framework .NET**, sviluppato praticamente di pari passo, segue una numerazione differente e la sua versione più recente, al momento della stampa, è la **4.6.1**.

Microsoft inoltre, che già forniva agli sviluppatori Windows l'ambiente di sviluppo integrato denominato Visual Studio, ha rilasciato da subito la versione Visual Studio .NET per adeguarlo al nuovo mondo, aggiornando costantemente anche tale ambiente e affiancandolo alle versioni di .NET susseguitesi nel tempo.
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 6, e di .NET Framework 4.6.1, utilizzando come ambiente di sviluppo la versione **Visual Studio 2015**.

A chi si rivolge il libro

Questo libro intende rivolgersi al lettore che si avvicina per la prima volta al mondo della programmazione, ma anche 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 sono infatti 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 o trovare risposte a domande e questioni più avanzate.
Inoltre, per chi, come il sottoscritto, vive e lavora nel mondo della programmazione da decenni e che quindi affronta l'argomento con ancora maggiore passione, ho cercato di trovare e fornire degli spunti e delle curiosità legate a C# e .NET nel diramarsi delle pagine e dei capitoli, facendo notare qual è stata l'evoluzione del linguaggio lungo le sue varie versioni e indicando in quale di esse è stata introdotta ogni nuova funzionalità o caratteristica.

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, partendo quindi dalle basi del linguaggio **C# 6** fino ad arrivare ai concetti più complessi, permettendo di padroneggiare così ogni argomento che riguardi la programmazione .NET, anche quelli non trattati in questo testo.
La prima parte del libro, costituita dai primi cinque capitoli, introduce il framework .NET e le caratteristiche del linguaggio C# puro, iniziando dalle parole chiavi, dalla sintassi con cui si scrivono i programmi, introducendo i tipi fondamentali del framework

.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 dei suoi componenti, come il CLR e la Framework Class Library, 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#.

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 oppure l'ambiente integrato Visual Studio. 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 mostra 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#, perciò 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 generics.

Il **Capitolo 10** tratta una questione fondamentale di C#, cioè quella della programmazione a eventi e dei metodi di gestione degli stessi, e di argomenti strettamente legati, come quello dei delegate, dei metodi anonimi e delle espressioni lambda.

Avanzando lungo i capitoli si copriranno tutte le sfaccettature del linguaggio C#, introdotte nelle sue varie versioni. Il **Capitolo 11** include quindi anche materie 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 fatta una panoramica della .NET Compiler Platform, o Roslyn, mostrando come utilizzare i servizi messi a disposizione del compilatore nelle applicazioni o scrivere estensioni di Visual Studio.
Nell'ultimo, il **Capitolo 17**, tramite lo sviluppo di semplici esempi in diversi ambiti, 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 espressioni 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 e sull'utilizzo da codice gestito di funzioni native, per mezzo dei servizi P/Invoke.
L'**Appendice C** contiene le soluzioni 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.

Esempi pratici e capitoli bonus

Ogni capitolo del libro contiene esempi pratici che potete scrivere e compilare autonomamente, sia utilizzando il compilatore a riga di comando csc, sia utilizzando l'ambiente di sviluppo Visual Studio, in particolare la versione 2015; anche le versioni precedenti sono adeguate se non avete intenzione di utilizzare le caratteristiche del linguaggio introdotte con C# 6 (ma se state leggendo questo libro probabilmente vorrete farlo!).
Per chi non fosse provvisto di una licenza professionale di Visual Studio 2015 non c'è alcun problema: come avremo modo di vedere già dai primi capitoli, è disponibile una versione denominata Community liberamente scaricabile da Internet, 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 eventuali capitoli bonus che, per limiti di spazio o per aggiornamenti al 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 quasi 15 anni. In tale ambito il riconoscimento forse più importante è la nomina a Microsoft MVP per .NET da gennaio 2015 e MVP per Visual Studio e Tecnologie di Sviluppo da gennaio 2016.
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, nel 2014, la prima edizione di "Programmare con C# 5, guida completa".
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 diventando consulente e freelance su progetti software di ogni dimensione e tipo.
Fa parte dello user group siciliano Orange Dot Net (http://www.orangedotnet.org) e partecipa spesso a conferenze ed eventi in qualità di speaker, trattando sempre argomenti legati allo sviluppo software.
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, i messaggi e le 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 sia 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.

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 un altro dispositivo e che quindi hai già riposto fiducia nei miei confronti (magari per la seconda volta, se hai già letto la prima edizione!).

Grazie **Nonna Concetta**, che mi protegge sulla stella più bella che c'è.

Infine, e come sempre, grazie a **mia moglie Caterina**, che mi sopporta, mi incoraggia e mi sostiene istante per istante e che, durante gli ultimi tempi di lavoro e revisione, ha portato, spesso faticosamente, in grembo la nostra prima bimba **Matilda**, che adesso è senza dubbio la cosa più importante della nostra vita. `while(true) {ViAdoro();}`

1

C# e la piattaforma .NET

Il .NET Framework è un ambiente di sviluppo ed esecuzione di applicazioni, introdotto da Microsoft all'inizio degli anni 2000, insieme a C#, linguaggio di programmazione multiparadigma e principalmente orientato agli oggetti.

La piattaforma .NET nasce alla fine degli anni Novanta, quando Microsoft inizia a lavorare a un nuovo e rivoluzionario modello di sviluppo e programmazione dei propri (ma non solo) 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.
Insieme al nuovo framework, che assume il nome definitivo di **.NET Framework** quando viene ufficialmente annunciata la sua prima beta pubblica (il 13 novembre 2000), viene progettato un nuovo linguaggio di programmazione, anch'esso denominato con un nome provvisorio, **COOL** (*C-Like Object Oriented Language*), per poi essere battezzato con l'ormai noto nome di **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 sta 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

distribuite e interconnesse, appartenenti al mondo desktop, ma anche al Web e, negli ultimi anni, al mondo dei dispositivi mobili come smartphone e tablet.
Essendo quindi C# così fortemente e quasi indissolubilmente legato alla piattaforma .NET, 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 del .NET Framework costituisce un prerequisito fondamentale della programmazione in C#.
Non è infatti minimamente immaginabile pensare a C# come a un linguaggio di programmazione a sé stante e slegato da .NET, anche se sia il linguaggio sia l'ambiente di esecuzione dei programmi, seppur inventati da Microsoft, sono divenuti degli standard ECMA e ISO/IEC, che chiunque può liberamente implementare su qualunque piattaforma; Microsoft ha inoltre reso open source importanti parti sia del framework .NET e delle librerie di base, sia il compilatore di C# e relativi servizi.

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 spesso accade, 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 costituivano l'interfaccia di programmazione del sistema operativo in linguaggio C, oppure fare uso delle Microsoft Foundation Classes (MFC) in C++; altri ancora sceglievano *COM*, la modalità di sviluppo indipendente dal linguaggio scelto per creare oggetti e componenti, che forse può essere considerata il predecessore di .NET, almeno negli intenti e negli scopi.
Dalla prima versione di Windows, la 1.0 rilasciata nel 1985, 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 esaustività, sigle e nomi come Win32, COM, ATL, DDE, ActiveX, MFC, Visual Basic ecc.
Arrivati alla fine degli anni Novanta, 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 a cui 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 non è legato ad alcun sistema operativo in particolare, anche se Microsoft ha sempre rilasciato le proprie versioni per l'esecuzione su sistemi **Windows** e **Windows Mobile/Windows Phone**. Oggi, la visione strategica più recente e aggiornata di Microsoft ha dato vita, sotto il nome di **.NET 2015**, a un framework all'insegna del cross platform e dell'open source, che consentirà il funzionamento ottimale anche su sistemi Linux e Mac.
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 su sistemi **Linux** e **MacOS X** e che comprende anche il compilatore C# e strumenti di sviluppo come *MonoDevelop*.
Ulteriori implementazioni consentono inoltre 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).
Oltre a non essere limitato ad alcun sistema operativo, .NET non ha vincoli 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, naturalmente fra quelli 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.

Definizione di .NET

.NET Framework è una piattaforma per lo sviluppo e un ambiente di esecuzione che fornisce vari servizi alle applicazioni in esecuzione al suo interno.
Esso è costituito da due componenti principali: il **Common Language Runtime** (CLR), ovvero il motore di esecuzione che gestisce l'intero ciclo di esecuzione delle applica-

zioni, e la **libreria di classi di base** del .NET Framework, che fornisce una raccolta di codice testato e riutilizzabile dagli sviluppatori nelle proprie applicazioni.
Rispettando gli obiettivi che il team .NET si era posto, il framework possiede le seguenti caratteristiche di base:

- fornire una piattaforma di sviluppo moderna e orientata agli oggetti;
- interoperabilità fra diversi linguaggi di programmazione; i compilatori dei linguaggi utilizzabili in .NET generano un codice intermedio, denominato Common Intermediate Language (CIL), che, a sua volta, viene compilato in fase di esecuzione dal 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;
- multitarget in quanto è possibile creare delle librerie portabili che funzionano su diverse piattaforme come Windows, Windows Phone, Xbox;
- un sistema di tipi comune, infatti 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) accessibili e utilizzabili da qualunque linguaggio che supporti il .NET Framework;
- un'estesa libreria di classi di base che i programmatori hanno a disposizione; possono quindi utilizzare una libreria di tipi e relativi membri facilmente accessibili dalla libreria di classi .NET Framework, anziché dover scrivere grandi quantità di codice per gestire operazioni comuni di programmazione di basso livello. Inoltre .NET Framework include delle librerie specifiche per particolari aree applicative, per esempio 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;
- un ambiente di esecuzione dalle performance elevate e ottimizzate per ogni piattaforma su cui esso è supportato;
- compatibilità di versione; infatti, in generale, un'applicazione sviluppata per una data versione del .NET Framework continuerà a funzionare su tutte le versioni successive;
- esecuzione side-by-side, cioè la possibile coesistenza di diverse versioni dell'ambiente di esecuzione di .NET sullo stesso computer, in maniera che ogni applicazione o ogni sua versione particolare 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.

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 dell'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 per riuscire a ottenere un livello elevato di conoscenza e messa in pratica efficace delle sue caratteristiche.

Il linguaggio C#

C# si pronuncia *C sharp* (oppure in inglese come se si leggessero le parole *see sharp*) ed è un linguaggio di programmazione orientato agli oggetti, ma con caratteristiche che gli permettono di essere definito multiparadigma, semplice e moderno, type-safe e dall'utilizzo non limitato o vincolato ad alcun ambito applicativo.
Esso affonda le sue radici nella famiglia di linguaggi che fanno capo al C e la sua sintassi lo rende molto simile a C++ e Java, tanto che originariamente il creatore di Java, *James Gosling*, dichiarò apertamente che C# era una semplice imitazione del suo linguaggio, anzi che C# era Java privato di affidabilità, produttività e sicurezza; in effetti non era possibile negare, all'epoca, che la sintassi dei due linguaggi fosse molto simile e che il .NET Framework (o come vedremo meglio a breve il CLR) fosse un concetto analogo alla *JVM* (*Java Virtual Machine*), l'ambiente di esecuzione dei programmi Java.

NOTA

Perché si chiama C#? In quanto linguaggio derivato dal C, il nome C# deriva dalla combinazione di C con il simbolo diesis preso dalla notazione musicale, che indica che la nota a cui si riferisce va alzata di un semitono, a intendere un linguaggio che è un passo superiore a C. Secondo altre interpretazioni il simbolo # è invece la combinazione di quattro +, quindi un modo per scrivere C++++. Per gli amanti delle curiosità: il simbolo corretto del diesis è ♯, che in inglese si legge *sharp* ed è diverso dal simbolo del cancelletto #. Ricordo ancora con un certo divertimento qualcuno che si ostinava a chiamare il linguaggio *C cancelletto* o *C diesis* (se siete fra questi ora non avete più motivo per farlo)!

Dal momento della sua nascita e quindi dalla sua prima versione, esso si è però costantemente evoluto, introducendo importanti novità in ognuna delle successive revisioni, che hanno aggiunto a C# anche caratteristiche prese da linguaggi funzionali e dinamici.

NOTA

C# è stato standardizzato dall'ente ECMA International come ECMA-334 standard e da ISO/IEC con lo standard ISO/IEC 23270:2006. Il compilatore di C# per il .NET Framework realizzato da Microsoft è un'implementazione che rispetta i due standard. Il file pdf dello standard ECMA è liberamente scaricabile all'indirizzo http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-334.pdf. La versione C# 6 sfrutta invece la più recente implementazione open source del compilatore, chiamata in codice Roslyn, e che fa parte della .NET Compiler Platform.

Il creatore del linguaggio C# è *Anders Hejlsberg*, ingegnere software danese, leggenda vivente del mondo della programmazione. Hejlsberg vanta un'esperienza trentennale in Borland, dove ha creato il compilatore Turbo Pascal e il suo successore, Borland Delphi. Nel 1996 approda in Microsoft, dove inizialmente si occupa del *J++* (versione Microsoft del linguaggio Java) e delle Windows Foundation Classes, per poi divenire l'architetto capo di C#.

Storia di C# e .NET

Fornire una sorta di cronistoria delle varie versioni di .NET e C# non è una questione di pura curiosità, ma serve a comprendere le correnti evolutive che hanno portato alla versione attuale di .NET e C#. Quindi verrà fatto ora un breve riepilogo delle release che si sono susseguite in questi anni.

C# viene annunciato al pubblico per la prima volta nel luglio del 2000 alla Professional Developer Conference, presentato insieme al .NET Framework e alla prima versione .NET di Visual Studio. In quell'occasione una versione beta viene rilasciata anche ai partecipanti alla conferenza.

La prima beta pubblica viene poi rilasciata il 13 novembre dello stesso anno, mentre la versione RC (Release Candidate) è annunciata da Bill Gates nell'ottobre del 2001.

La prima versione definitiva del .NET Framework, la **1.0**, vede quindi la luce nel gennaio del 2002, seguita dopo poche settimane da **Visual Studio .NET**: è la prima versione dell'IDE che permette di sviluppare applicazioni per il nuovo ambiente di esecuzione.

Nell'aprile del 2003 il .NET Framework viene aggiornato alla versione **1.1**, mentre C# passa direttamente alla versione **1.2**, l'unica versione minore fino adesso rilasciata.

Nella stessa occasione anche l'ambiente di sviluppo viene aggiornato, assumendo la denominazione di **Visual Studio .NET 2003**.
Per la versione **2.0**, sia di .NET sia di C#, bisogna invece attendere il 2005. Tale versione introduce numerosi e importanti cambiamenti e segna anche la direzione di una rotta evolutiva totalmente differente da quella del concorrente Java. Fra le novità troviamo i *generics*, le *classi parziali* e i tipi *nullable*.
Sempre nello stesso anno viene rilasciato **Visual Studio 2005**, che perde il suffisso .NET in quanto ormai implicitamente presente.
.NET **3.0** (nome in codice *WinFX*), rilasciato nel 2006, è una versione incrementale rispetto alla precedente, che non apporta cambiamenti alla struttura di base, rimasta uguale alla 2.0, ma introduce dei nuovi componenti di contorno (*WPF*, *WCF*, *WF* e *Cardspace*).
Alla fine del 2007, il .NET Framework passa alla versione **3.5** mentre le specifiche di C# sono aggiornate alla versione **3.0**, che introduce per la prima volta delle caratteristiche prettamente funzionali, come le espressioni *lambda* e *LINQ*. Visual Studio invece viene rilasciato nella versione **2008**.
Nell'aprile 2010, .NET e C# passano alle rispettive versioni **4.0**, mentre viene rilasciato anche **Visual Studio 2010**. Gli aggiornamenti principali di C# 4.0 sono costituiti dall'introduzione di caratteristiche tipiche di un linguaggio dinamico.
Nel febbraio 2012 il .NET Framework arriva alla versione **4.5** mentre Visual Studio viene aggiornato alla versione **2012**. Nell'agosto dello stesso anno vengono pubblicate le specifiche di **C# 5.0**, che includono come caratteristica fondamentale il paradigma di programmazione *asincrona*.
La Tabella 1.1 riepiloga in maniera concisa tale evoluzione lungo gli anni.

Tabella 1.1 - Storia delle versioni di .NET, C# e Visual Studio.

Anno	.NET Framework	C#	Visual Studio
2002	1.0	1.0	Visual Studio .NET 2002
2003	1.1	1.2	Visual Studio .NET 2003
2005	2.0	2.0	Visual Studio 2005
2006	3.0	2.0	Visual Studio 2005
2007	3.5	3.0	Visual Studio 2008
2010	4.0	4.0	Visual Studio 2010
2012	4.5	5.0	Visual Studio 2012
2013	4.5.1 4.5.2	5.0	Visual Studio 2013
2015	4.6	6.0	Visual Studio 2015

La versione **4.6** del .NET Framework e l'edizione **2015** di Visual Studio, rilasciate il 20 luglio 2015, hanno apportato diverse novità in termini di innovazione, di produttività e di nuove funzionalità messe a disposizione degli sviluppatori di qualunque tipologia di software: desktop, Web, mobile, cloud e così via.
Contemporaneamente il linguaggio C# arriva alla versione **6.0** e, naturalmente, l'IDE e il framework la supportano pienamente.
Questo testo è quindi perfettamente aggiornato e allineato e descrive tutte le ultime caratteristiche di C#, utilizzando l'ambiente di sviluppo Visual Studio 2015 per la stesura ed esecuzione degli esempi pratici. Notate che per l'utilizzo della versione 6.0 di C# è necessario utilizzare proprio Visual Studio 2015, perché integra appunto il nuovo compilatore del linguaggio.
Quando fra qualche anno verranno apportate novità al linguaggio in se stesso, nessuna paura, basterà ripartire da qui e studiare solo le nuove caratteristiche (magari acquistando l'edizione aggiornata di questo libro!).

NOTA

Se siete curiosi di conoscere cosa bolle in pentola e quali sono le caratteristiche oggetto di studio e di valutazione per una futura inclusione nelle specifiche del linguaggio C# (e magari se desiderate anche contribuire) potete seguire le pagine e le discussioni ufficiali su GitHub: https://github.com/dotnet/roslyn/wiki.

Versioni e dipendenza

Un componente fondamentale di .NET è il **CLR** (Common Language Runtime), cioè il suo ambiente di esecuzione, che vedremo più in dettaglio nei prossimi paragrafi.
Ogni versione di .NET Framework consiste quindi di una versione del CLR (oltre che di un insieme di librerie di base), ma il suo numero di versione non è mai andato di pari passo con la versione del framework.
Per esempio il CLR 2.0, una volta introdotto con la versione 2.0 del .NET Framework, è rimasto immutato anche per le versioni 3.0 e 3.5 del framework stesso (non esiste alcun CLR 3.x); ciò perché queste ultime due versioni costituiscono dei rilasci incrementali o aggiornamenti, che hanno aggiunto nuove librerie e funzionalità di alto livello messe a disposizione degli sviluppatori, mantenendo lo stesso ambiente di esecuzione precedente.
.NET Framework 4.0 introduce invece il CLR 4.0, mentre con il rilascio delle versioni seguenti di .NET, 4.5 e successive, il CLR non ha subìto aggiornamenti (vale a dire che il numero di versione principale è sempre 4.0).

NOTA

Il numero di versione di C# non coincide con quello del .NET Framework (vedere la Tabella 1.1), ma ciò non deve sorprendere: C# è un linguaggio, .NET è un framework, mentre il CLR è un suo componente fondamentale; tuttavia non si può negare che la politica di numerazione delle versioni di .NET, del CLR e del linguaggio C# è stata spesso molto confusionaria.

Ogni versione di .NET è in generale compatibile con la precedente, in maniera da poter continuare a eseguire applicazioni scritte per una data versione di .NET anche su ogni versione successiva; quando non è così, le versioni di .NET sono comunque installabili side-by-side.

Installazione *side-by-side* significa che sullo stesso sistema operativo è possibile installare diverse versioni di .NET una affiancata all'altra. In questo modo ogni applicazione verrà eseguita sull'ambiente .NET appropriato e con le librerie adeguate.

Nelle più recenti versioni del sistema operativo Windows, il .NET Framework è inoltre già presente senza necessità di installazioni aggiuntive. Per esempio, su Windows 10 come impostazione predefinita è già presente il .NET Framework 4.6 e 3.5, su Windows 8.1 è presente la versione 4.5.1, mentre il 3.5 può essere abilitato direttamente dal pannello di controllo.

Potete verificare le versioni di .NET installate sul vostro sistema in differenti modi, per esempio esplorando la directory di installazione predefinita, in genere C:\Windows\Microsoft.NET\Framework.

La Figura 1.1 mostra il contenuto della cartella su un sistema con diverse versioni installate fino alla 4.5 (non è un errore, non c'è alcuna cartella 4.5.xxx perché, come già detto, il .NET 4.5 è installato come aggiornamento sul posto).

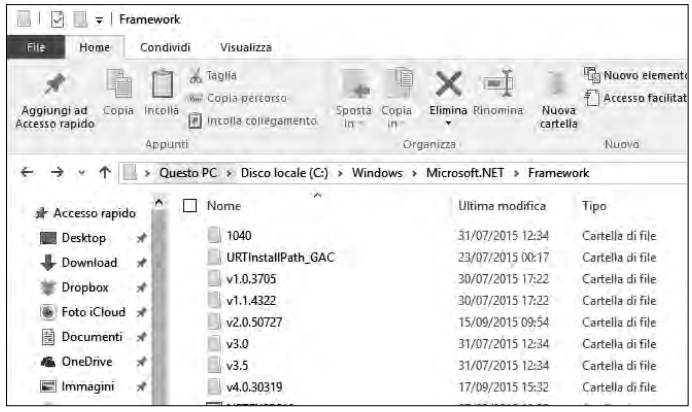

Figura 1.1 – Directory di installazione di .NET Framework.

Una modalità più precisa per conoscere le versioni di .NET presenti su un computer, o per stabilire la versione esatta, è quella di interrogare il registro. Se siete utenti abbastanza esperti potete quindi aprire l'editor di registro (regedit.exe) e verificare il valore `Release` della chiave `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\NET Framework Setup\NDP\v4\Full`. La Figura 1.2 mostra il contenuto di tale chiave su una macchina che esegue Windows 10, e in tale caso il valore Release è pari a 393295 (393297 su altri sistemi operativi). Per verificare invece il numero di versioni del CLR presenti sul sistema, potete lanciare il comando `clrver` da un prompt dei comandi di Visual Studio (più avanti vedremo anche come aprire tale prompt). Il risultato stampato sulla stessa macchina è il seguente:

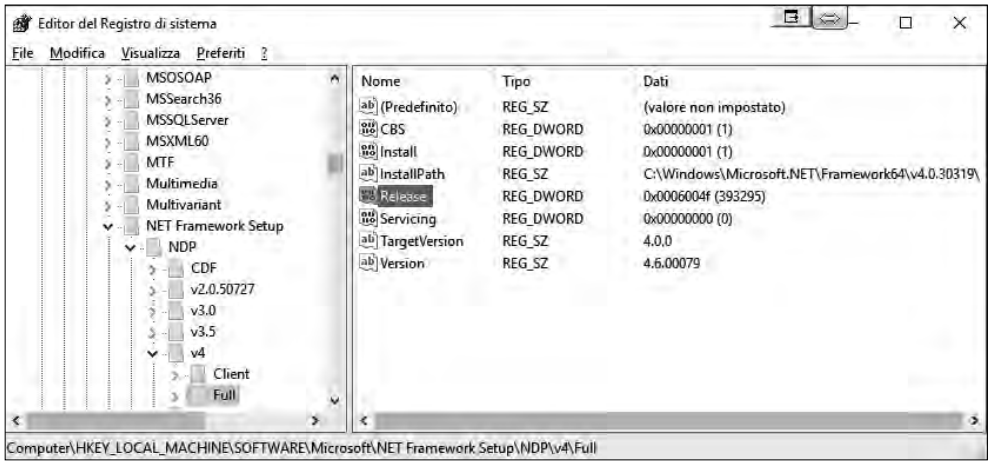

Figura 1.2 – Chiavi del registro di Windows con le versioni di .NET Framework.

```
C:\Program Files (x86)\Microsoft Visual Studio 14.0> clrver

Microsoft (R) .NET CLR Version Tool  Version 4.6.81.0
Copyright (c) Microsoft Corporation.  All rights reserved.

Versions installed on the machine:
v2.0.50727
v4.0.30319
```

Quindi sono presenti in questo caso sia il CLR 2.0 che il CLR 4.0.

NOTA

Se siete abbastanza esperti da sapere scrivere e compilare un programma in C#, potete invece utilizzare il metodo `RuntimeEnvironment.GetSystemVersion` per ottenere la versione del CLR in cui è in esecuzione il programma stesso.

.NET Framework 4.6 è compatibile con le applicazioni precedenti compilate con le versioni di .NET Framework 1.1, 2.0, 3.0, 3.5, 4 e 4.5.1/4.5.2. In altre parole, le applicazioni e i componenti compilati con le versioni precedenti di .NET Framework funzioneranno su .NET Framework 4.6.

NOTA

Non è necessario installare versioni precedenti di .NET Framework o di CLR prima di installare la versione più recente.

La Figura 1.3 mostra una visione d'insieme degli aggiornamenti del CLR, con i vari rilasci del framework e le principali novità introdotte. Al momento della revisione finale di questo libro, la versione più recente del .NET Framework è la 4.6, che introduce vari miglioramenti alla piattaforma, nuove API e correzioni di bug.

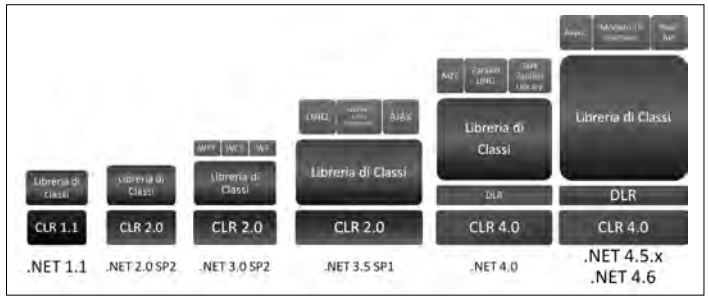

Figura 1.3 – Versioni del .NET Framework e novità introdotte.

.NET 2015

.NET è naturalmente un componente strategico per il mondo dello sviluppo software visto da Microsoft. Il 2015 è stato un anno di particolare innovazione, che ha portato una ventata di novità in questo mondo. Secondo questa nuova visione, l'intera piattaforma ha assunto il nome di .NET 2015 ed essa non riguarda solo le tecnologie di sviluppo vere e proprie, ma può essere considerata anche una nuova filosofia.
Il .NET Framework 4.6 fa parte di questa piattaforma, insieme a un nuovo framework denominato .NET Core 5.
La Figura 1.4 riassume quanto è oggi racchiuso sotto il termine .NET 2015.
.NET Framework è ancora naturalmente una parte fondamentale e verrà esaminato in maniera più approfondita nei prossimi paragrafi. Esso non è cross platform e può essere installato solo su Windows (anche se, come già detto, esistono altre implementazioni non Microsoft che permettono di eseguire applicazioni .NET su altri sistemi).

Al di sopra del .NET Framework, i suoi utilizzatori sono le applicazioni WPF e Windows Forms per il desktop, e ASP.NET per il Web.

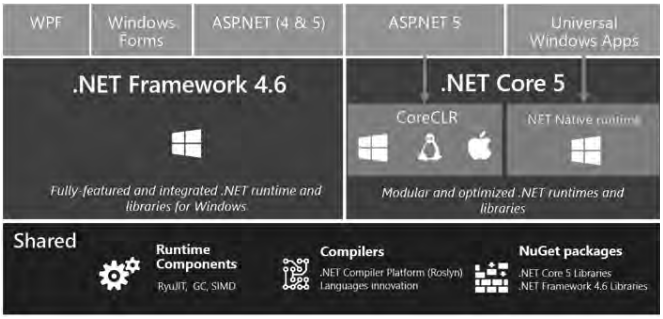

Figura 1.4 – .NET 2015.

.NET Core

.NET Core è una versione modulare del .NET Framework, di cui costituisce le fondamenta, e può essere utilizzato per lo sviluppo di applicazioni in un'ampia varietà di ambiti specifici.
La differenza principale con il fratello maggiore è che esso è completamente costituito da componenti che possono essere installati localmente come pacchetti insieme alle proprie applicazioni, utilizzando solo quelli necessari.
.NET Core consiste di CoreCLR, cioè il runtime o motore di esecuzione e le librerie di base, di CoreFX, cioè le librerie con le classi fondamentali per lo sviluppo, e del compilatore.
Infine, proprio per rendere e mantenere .NET Core un framework cross platform, quindi disponibile su diversi sistemi operativi (in particolare Microsoft supporta Windows, Linux e Mac OSX), esso è open source.

NOTA

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

Un altro runtime, appositamente pensato per eseguire le applicazioni Universal Windows (vale a dire quelle distribuite tramite Windows Store e che girano sul sistema operativo Windows 10), è denominato **.NET Native Runtime**. In questo caso le applicazioni sono automaticamente compilate in codice nativo e quindi ottimizzate per un'esecuzione ottimale sul sistema operativo Microsoft.

CoreCLR e .NET Native Runtime possono essere utilizzati direttamente dalle applicazioni, oppure costituire uno strato per degli utilizzatori di più alto livello. Per esempio la Figura 1.4 mostra come ASP.NET 5 utilizzi il CoreCLR (oppure il .NET Framework completo visto prima) mentre le Universal Windows Apps si basano sul .NET Native Runtime.

.NET Compiler Platform

La principale novità di C# 6 può essere considerata l'introduzione di un nuovo compilatore. La **.NET Compiler Platform**, nota anche con il nome in codice di **Roslyn**, è la piattaforma open source di compilazione e di strumenti di analisi dedicata ai linguaggi C# e VB.

L'idea alla base di questa piattaforma è quella di consentire l'accesso alle funzioni di compilazione come se fossero un insieme di servizi e non solo una scatola nera a cui dare in pasto del codice e ottenerne un programma eseguibile o una libreria compilata. Secondo questa filosofia, nota anche come *Compiler as a Service*, il compilatore stesso, o meglio la sua API (Application Programming Interface), potrà essere utilizzata nelle proprie applicazioni, per esempio per creare delle funzioni di analisi, diagnostica, generazione ed elaborazione del codice, come quelle fornite da Visual Studio.

NOTA

La pagina ufficiale e il codice sorgente della .NET Compiler Platform (o Roslyn) si trovano su GitHub al seguente url: https://github.com/dotnet/roslyn/.

Architettura di .NET Framework

Il .NET Framework è una piattaforma complessa e, al tempo stesso, una tecnologia che fornisce tutto il necessario per lo sviluppo e l'esecuzione di applicazioni eterogenee: desktop, mobile, Web, servizi e così via.

Il framework consiste di due componenti principali: il suo ambiente di esecuzione, detto CLR, e la libreria di classi, detta *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 1.5 mostra una visione di insieme dell'architettura a livelli di .NET.

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.

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 ancora 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.

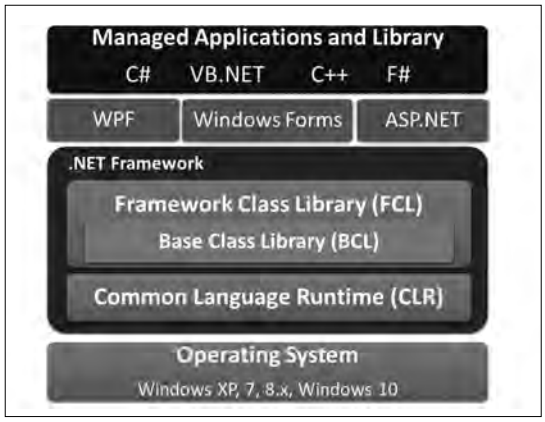

Figura 1.5 – Architettura a livelli del .NET Framework.

Per la scrittura del codice sorgente delle applicazioni può essere utilizzato uno qualunque dei linguaggi cosiddetti *.NET Enabled* (nella Figura 1.5 sono elencati solo i linguaggi supportati in maniera predefinita da Visual Studio 2015).

Common Language Runtime

Il 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 .NET Enabled, cioè 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.

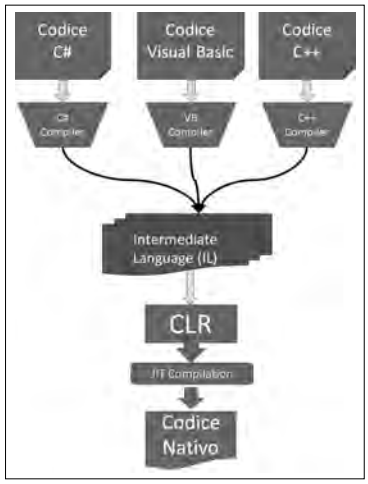

Figura 1.6 – Compilazione ed esecuzione in .NET.

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 il codice specifico della piattaforma: per esempio se esso è in esecuzione su una versione x86 di Windows esso produrrà istruzioni x86. 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.
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 gestione della memoria e degli 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 un'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 cosiddetto entry point, cioè un punto di ingresso 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 *risorse* opzionali (per esempio immagini, audio ecc.).
Grazie a tali metadati ogni assembly è completamente autodescrittivo 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 del .NET Framework su tale computer).

Figura 1.7 – Formato e contenuto di un assembly .NET.

La Figura 1.7 mostra il formato di un assembly .NET e il 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.

DLL Hell

Il concetto di assembly ha cercato di mettere fine al cosiddetto problema dell'inferno delle DLL (*DLL Hell*), rendendo anche notevolmente più semplice la distribuzione delle applicazioni.

Prima dell'arrivo di .NET, infatti, il problema di versionamento delle DLL e della loro condivisione provocava spesso malfunzionamenti nelle applicazioni installate, dovuti, per esempio, all'installazione di nuove versioni della stessa applicazione o, peggio ancora, di altre applicazioni che utilizzavano la stessa DLL in una versione differente.
Si immagini, per esempio, un'applicazione A che utilizza la DLL versione 1.0; successivamente viene installata l'applicazione B che utilizza la stessa DLL ma in versione 1.1 e sovrascrive la precedente. A questo punto A smette di funzionare per qualche strano motivo (probabilmente utilizzava una funzione che è stata modificata nella versione 1.1). Per sistemare le cose l'utente reinstalla l'applicazione A ripristinando la versione DLL 1.0. La conseguenza è che smette di funzionare l'applicazione B che richiede la DLL 1.1.
Immaginate ora questo scenario per decine di applicazioni: ecco il cosiddetto inferno delle DLL.
Uno degli obiettivi progettuali del team del .NET Framework era quello di trovare una soluzione definitiva alla problematica descritta. Per farlo .NET fornisce le seguenti funzionalità e caratteristiche:

- le applicazioni sono **auto-descrittive**, rimuovono la dipendenza dal registro, non impattano sul funzionamento di altre applicazioni installate e consentono una semplice e indolore disinstallazione;
- le informazioni relative alle **versioni** devono essere registrate e ricordate, in maniera che ogni applicazione conosca per esempio quale insieme di assembly e quale versione deve utilizzare;
- supporto per consentire l'esistenza di **molteplici versioni** dello stesso assembly **side-by-side** (cioè fianco a fianco) in maniera che ogni applicazione possa utilizzare la versione più appropriata. Il .NET Framework porta il concetto di installazione side-by-side ancora più avanti, consentendo l'installazione di più versioni di se stesso sullo stesso sistema, in maniera che più applicazioni che richiedono versioni diverse di .NET siano comunque eseguibili;
- **isolamento delle applicazioni** cioè la possibilità di rendere possibile l'esecuzione delle applicazioni in maniera indipendente dal sistema operativo su cui sono installate; per esempio è possibile distribuire tutti gli assembly richiesti da un'applicazione nella stessa directory e quindi senza complesse procedure di installazione e registrazione di componenti e DLL. Allo stesso tempo l'installazione o disinstallazione di altre applicazioni non influisce su tale applicazione isolata.

La Framework Class Library

Il .NET Framework fornisce una gigantesca libreria di assembly, contenente migliaia di oggetti (chiamati tipi) utilizzabili all'interno delle proprie applicazioni. Tale libreria è chiamata **Framework Class Library** (FCL).

È praticamente impossibile conoscere tutti i tipi messi a disposizione ed è anzi compito del programmatore imparare a cercare al suo interno le funzionalità di cui ha bisogno nello sviluppo di ogni particolare applicazione.
La libreria mette infatti a disposizione tipi per diverse tipologie di applicazioni (desktop, Web, web service, mobile e così via) e per diversi ambiti applicativi (input/output, database, XML ecc.), suddividendoli in una struttura gerarchica mediante il concetto di spazio dei nomi o namespace (vedere il Capitolo 2).
Un sottoinsieme contenente i tipi di base e non correlato ad alcuna particolare tecnologia è detto anche **Base Class Library** (BCL).
Un *tipo* è una rappresentazione di dati, quindi la BCL contiene, fra gli altri, i tipi che rappresentano dati fondamentali come i numeri.
La libreria è utilizzabile naturalmente da qualsiasi linguaggio .NET visto che, come abbiamo già accennato, gli assembly sono prodotti a partire da uno stesso linguaggio intermedio IL.
In realtà, a partire dalla versione 3.0 di .NET, è stata modificata la modalità di distribuzione e di raggruppamento dei tipi messi a disposizione dal framework.
Tale versione infatti ha introdotto nuovi insiemi di tipi, raggruppandoli in nuove librerie, ognuna dedicata a una particolare area di sviluppo:

- **WPF** (*Windows Presentation Foundation*) – contiene tipi per lo sviluppo di interfacce grafiche di nuova generazione, con un nuovo paradigma di sviluppo e disegno e un nuovo linguaggio di definizione dell'interfaccia, chiamato *XAML;*
- **WCF** (*Windows Communication Foundation*) – un insieme di tipi per realizzare servizi web e applicazioni che si scambiano dati in rete in maniera indipendente dal protocollo (http, tcp ecc.);
- **WF** (*Workflow Foundation*) – un framework per lo sviluppo di applicazioni basate su flussi di attività.

Con la versione 4.6 i tipi contenuti nella Framework Class Library e utilizzabili dagli sviluppatori sono circa quindicimila!

Common Type System

Data l'importanza fondamentale dei tipi all'interno della piattaforma .NET, Microsoft ha creato una specifica formale, detta **Common Type System** (CTS), nella quale viene descritto come sono definiti i tipi e come funzionano.
Tale specifica è comune ed è condivisa fra i vari linguaggi di programmazione supportati da .NET, quindi permette la piena interoperabilità fra applicazioni e librerie scritte in linguaggi diversi.

Prima dell'era .NET un problema abbastanza comune nello sviluppo di applicazioni scritte in tecnologie o linguaggi differenti, in particolare allo scopo di ottenerne l'interoperabilità, era lo scambio di dati.
Basta pensare che un tipo che rappresenta una sequenza di caratteri di testo, la stringa, era rappresentato e chiamato in maniera differente in quasi ognuno dei linguaggi utilizzati: char* in C, CString in MFC, BSTR in COM.
In .NET il tipo stringa è implementato all'interno della libreria di base, è denominato `System.String`, ed è lo stesso per ogni linguaggio, sia esso C#, Visual Basic, C++ o un altro linguaggio .NET.
Un altro esempio è quello del tipo che definisce i numeri interi che, per esempio, in alcuni linguaggi è dipendente dalla piattaforma e dal compilatore, quindi utilizzerà 16, 32, 64 bit e così via.
In .NET invece il tipo `int` ha sempre 32 bit, su qualunque sistema operativo, in qualunque linguaggio.
Per ottenere questo obiettivo il CTS stabilisce le regole del gioco sulla creazione dei tipi e la prima di queste regole è che ogni tipo ha una madre comune dalla quale eredita varie caratteristiche. La madre di tutti i tipi è la classe `System.Object`.

Common Language Infrastructure

Come parte della strategia di Microsoft, la **Common Language Infrastructure** (CLI) è una specifica che consente a un programma scritto in un qualsiasi linguaggio supportato da .NET di essere eseguito su un qualunque sistema operativo, utilizzando un ambiente di esecuzione comune (cioè il già citato CLR).
CLI è stato approvato come standard dall'ECMA (ECMA-335) e comprende e specifica:

- un linguaggio comune (CLS, Common Language Specification) e le regole per le interoperabilità fra linguaggi;
- un insieme comune di tipi (CTS, Common Type System);
- una serie di informazioni sulla struttura di un programma, indipendenti dal linguaggio, per permettere a programmi scritti in linguaggi diversi di comunicare fra loro;
- un ambiente di esecuzione virtuale comune, che esegue i programmi.

Si faccia attenzione al fatto che il CLI non è un componente o un'implementazione, ma solo una specifica, che può essere implementata in modi diversi.
Il .NET Framework è l'implementazione proprietaria di Microsoft della specifica CLI, che comprende il CLR e un'ampia collezione di librerie, risorse e strumenti di sviluppo.
.NET Core è un'implementazione open source che implementa lo standard ECMA 335.

Il progetto Mono è invece un'iniziativa promossa e sponsorizzata da Novell per creare una versione open source per i sistemi UNIX della piattaforma .NET.

Strumenti di programmazione

Per scrivere un qualunque programma in C# è necessaria una dotazione minima costituita da un editor di testo, con il quale si scriverà e salverà il cosiddetto codice sorgente. Per trasformare poi il codice sorgente in un programma eseguibile sarà necessario darlo in pasto a un compilatore del linguaggio, nel nostro caso un compilatore C#.
Il compilatore standard fornito insieme al .NET Framework è chiamato *C# compiler* ed è utilizzabile dal prompt dei comandi lanciando l'eseguibile **csc.exe** (vedere Paragrafo "Developer Command Prompt" più avanti).
Così però la produttività non sarebbe di certo il massimo e quindi, per costruire applicazioni complesse in maniera molto più efficiente, sarà necessario abbandonare il blocco note e il prompt dei comandi e utilizzare appositi ambienti di sviluppo integrato, detti anche IDE (Integrated Development Environment), il cui rappresentante più noto nel mondo .NET è senza dubbio *Visual Studio* (del quale abbiamo già avuto modo di elencare le varie versioni, rilasciate in parallelo a quelle di .NET Framework e C#).
Nel prossimo capitolo si vedrà come compilare i primi esempi di codice utilizzando il compilatore a riga di comando **csc.exe**, in maniera da comprendere più a fondo il processo che porta a ottenere degli assembly .NET, ma nel resto del libro utilizzeremo quasi esclusivamente Visual Studio come ambiente di sviluppo e compilazione.
Essendo però Visual Studio uno strumento fin troppo grande e potente, soprattutto nella prima parte del libro e per chi è alle prime armi, il suo utilizzo sarebbe paragonabile allo sparare a una mosca con un cannone; infatti, fra le altre possibilità rapide per testare delle semplici istruzioni o programmi C#, vi è anche quella di utilizzare qualche semplice programmino che ci eviti sia di dover salvare un file di testo per poi darlo in pasto a csc.exe, sia di dover creare appunto un progetto Visual Studio.
Fra questi programmi, vi suggerisco *Linqpad*, disponibile anche in una versione gratuita sul sito www.linqpad.net, che supporta anche C# 6 e permette di eseguire programmi o istruzioni e osservarne rapidamente il risultato.
Un'altra possibilità, che sfrutta la nuova piattaforma di compilazione ed è disponibile con l'Update 1 di Visual Studio 2015, è quella di utilizzare la finestra **C# Interactive**, attivabile dal menu View -> Other Windows. Questo strumento è un editor REPL (read-eval-print-loop), all'interno del quale è possibile scrivere una qualunque espressione o istruzione C# ed eseguirla premendo Invio. Ulteriori informazioni e documentazioni sono disponibili a questo link, https://github.com/dotnet/roslyn/wiki/Interactive-Window.

NOTA

L'editor interattivo della finestra C# Interactive è disponibile anche da linea di comando, fuori da Visual Studio. Basta aprire un prompt dei comandi di sviluppo (vedi più avanti nel capitolo, "Developer Command Prompt") e digitare il comando `csi (C# interactive)`" (CSharp Interactive) per iniziare una nuova sessione.

.NET Framework SDK

Per poter sviluppare in C# o sulla piattaforma .NET in generale non è necessario acquistare Visual Studio o altre licenze Microsoft, come molti penseranno.

Il **.NET Framework SDK** (Software Development Kit) è incluso all'interno del **Windows SDK**, contiene tutto ciò che è necessario per lo sviluppo in .NET, ed è scaricabile gratuitamente insieme a decine di altre risorse, librerie, componenti e strumenti dal sito ufficiale http://msdn.microsoft.com/netframework.

Installando Visual Studio 2015, comunque, non sarà necessario nient'altro per poter sviluppare in C#, in quanto la procedura di setup dell'ambiente di sviluppo provvederà anche a installare l'SDK del .NET Framework.

Visual Studio 2015

Per sviluppare in maniera professionale ed efficiente un IDE come Microsoft Visual Studio costituisce una scelta quasi obbligata (anche se non l'unica).

Oltre alle varie versioni a pagamento, Visual Studio 2015 è disponibile anche nella versione Community, che è gratuita per singoli sviluppatori o piccoli team.

Per scrivere gli esempi di questo libro è stato utilizzato Visual Studio Professional 2015, ma se non avete la possibilità di acquistarlo, potete tranquillamente installare Visual Studio 2015 Community per iniziare a sviluppare in C#, ma anche per poter sviluppare applicazioni per il desktop, per il Web, per servizi cloud, per le Universal App di Windows 10, per Windows Phone, o anche per Android e iOS.

Tutto quello che imparerete a eseguire all'interno di Visual Studio 2015 Community sarà utilizzabile anche all'interno delle versioni più avanzate.

Nel seguito del libro quindi, quando ci riferiremo all'ambiente Visual Studio 2015 Community, lo chiameremo semplicemente Visual Studio 2015 o anche solo Visual Studio.

Visual Studio 2015 supporta lo sviluppo di applicazioni in uno dei linguaggi C#, Visual Basic, C++ e F#.

Nel prossimo capitolo verrà fornita un'ampia panoramica dell'ambiente di sviluppo Visual Studio 2015, a partire dalla composizione della sua interfaccia grafica, e saranno esposte le varie funzionalità messe a disposizione dello sviluppatore.

Developer Command Prompt

Dopo aver installato il .NET Framework SDK, oppure Visual Studio 2015, anche in versione Community, per aprire un prompt dei comandi pronto alla compilazione, cioè con la variabile di ambiente PATH contenente tutti i percorsi necessari, basta avviare il cosiddetto Developer Command Prompt.
Su Windows 7 è avviabile, in genere, dal menu **Start -> Programmi -> Microsoft Visual Studio 201x -> Visual Studio Tools**, mentre su Windows 8.x o Windows 10 basta avviare la ricerca (**Tasto Windows+Q**) e digitare **Developer Command Prompt for VS201x**.
Nel prossimo capitolo si tornerà sull'argomento, fornendo qualche dettaglio in più.
Da un prompt dei comandi per sviluppatori così avviato è possibile lanciare la compilazione di un programma C# semplicemente digitando il comando **csc** seguito dal percorso (o solo il nome se ci si trova già nella cartella) del file contenente il programma da compilare (csc è l'acronimo di *CSharp Compiler*).
Se volete conoscere le opzioni del compilatore basta digitare il seguente comando nello stesso prompt:

```
csc –?
```

Se tutto va bene e se il .NET Framework SDK è installato correttamente, la prima riga stampata dopo aver premuto il tasto Invio riporterà delle informazioni sulla versione del compilatore, per esempio:

```
Compilatore Microsoft (R) Visual C# versione 1.0.0.50618
```

Nel prossimo capitolo si utilizzerà il compilatore csc per creare i primi esempi di programmi in C#.

Riepilogo

Per riuscire a diventare sviluppatori esperti di un linguaggio basato su .NET come C# è praticamente un obbligo capire l'architettura e il funzionamento del .NET Framework. In questo primo capitolo quindi è stata esposta una panoramica di .NET e dei suoi componenti, come il CLR e la Framework Class Library, mostrando anche i concetti di base della compilazione ed esecuzione dei programmi scritti in C#. Infine abbiamo elencato gli strumenti di programmazione che utilizzeremo nel corso del libro e in ogni attività di sviluppo che dovremo affrontare come sviluppatori C#. Nei prossimi capitoli, alla fine di ognuno troverete delle domande di riepilogo, per mettervi alla prova e misurare il vostro grado di comprensione.