Introduzione

Questo è soprattutto un libro sulla Dependency Injection (DI). È anche un libro su .NET, ma molto meno. Sebbene per gli esempi di codice venga utilizzato il linguaggio C#, gran parte della discussione in questo libro può essere facilmente applicata ad altri linguaggi e piattaforme. In effetti, abbiamo imparato molti dei principi e dei modelli trattati leggendo libri nei quali per gli esempi sono stati usati i linguaggi Java o C++.
La Dependency Injection è un insieme di pattern e principi. È un modo di pensare e progettare il codice, più che una tecnologia. Lo scopo ultimo dell'utilizzo della Dependency Injection è quello di creare software gestibile all'interno del paradigma a oggetti. I concetti usati in questo libro si riferiscono tutti alla programmazione a oggetti. Il problema affrontato dalla Dependency Injection (la manutenibilità del codice) è universale, ma la soluzione proposta è data nell'ambito della programmazione a oggetti in linguaggi a tipi statici: C#, Java, Visual Basic .NET, C++ e così via. Non è possibile applicare la Dependency Injection alla programmazione procedurale, e potrebbe non essere la soluzione migliore nei linguaggi funzionali o dinamici.
Di per sé, la Dependency Injection è solo una piccola cosa, ma è strettamente correlata con un ampio complesso di principi e modelli per la programmazione a oggetti. Sebbene il libro si concentri sulla Dependency Injection dall'inizio alla fine, discute anche molti di questi altri argomenti, alla luce della prospettiva specifica della Dependency Injection. L'obiettivo del libro non è solo quello di insegnarvi le specifiche dalla Dependency Injection: l'obiettivo è quello di migliorare le vostre competenze come programmatori a oggetti.

A chi è rivolto questo libro?

Sarebbe molto allettante affermare che questo è un libro rivolto a tutti gli sviluppatori .NET. Ma la comunità .NET oggi è vasta e comprende sviluppatori che lavorano con applicazioni web, desktop, smartphone, RIA, integrazioni, automazione d'ufficio, sistemi di gestione dei contenuti e perfino giochi. Sebbene .NET sia una piattaforma a oggetti, non tutti questi sviluppatori scrivono codice a oggetti.
Questo è un libro sulla programmazione a oggetti, quindi come minimo i lettori dovrebbero essere interessati all'argomento e capire che cos'è un'interfaccia. Anche alcuni anni di esperienza professionale e conoscenza dei pattern di progettazione o dei principi

SOLID saranno sicuramente utili. In tutta onestà, non ci aspettiamo che i principianti si avvantaggino molto dal libro: è principalmente rivolto a sviluppatori esperti e architetti software.
Gli esempi sono tutti scritti in C#, quindi i lettori che impiegano altri linguaggi .NET devono essere in grado di leggere e comprendere il linguaggio C#. Anche i lettori che hanno familiarità con i linguaggi a oggetti non .NET, come Java e C++, possono trovare utile il libro, perché il contenuto specifico della piattaforma .NET è relativamente modesto. Personalmente, leggiamo molti libri di modelli con esempi in Java e ne ricaviamo molto, quindi speriamo che sia vero anche il contrario.

I contenuti

I contenuti di questo libro sono divisi in quattro Parti. Idealmente, vorremmo che prima lo leggeste dall'inizio alla fine e successivamente lo usaste come riferimento, ma potreste avere altre priorità. Per questo motivo, la maggior parte dei capitoli è scritta in modo che possiate immergervi subito e iniziare a leggere da quel punto.
La Parte I è la principale eccezione. Contiene un'introduzione generale alla Dependency Injection ed è probabilmente meglio leggerla da cima a fondo. La Parte II è un catalogo di pattern e cose simili, mentre la Parte III, più ampia, esamina la Dependency Injection da tre diverse angolazioni. La Parte IV è un catalogo di tre librerie di *container di Dependency Injection*.
Ci sono molti concetti interconnessi e, poiché li introduciamo la prima volta che ci è sembrato naturale, ciò significa che spesso menzioniamo i concetti prima ancora di averli introdotti formalmente. Per distinguere questi concetti universali dai termini più locali, utilizziamo il corsivo per farli risaltare. Tutti questi termini sono inoltre brevemente definiti nel Glossario, che contiene anche i riferimenti per una descrizione più estesa.
La Parte I è un'introduzione generale alla Dependency Injection. Se non sapete che cosa sia la Dependency Injection, questo è il giusto punto di partenza; ma anche se sapete di che cosa si tratta, potreste voler familiarizzare con i contenuti presentati nella Parte I, poiché definisce gran parte del contesto e della terminologia utilizzata nel resto del libro. Il Capitolo 1 discute gli scopi e i vantaggi della Dependency Injection e fornisce una panoramica generale. Il Capitolo 2 contiene un esempio ampio e piuttosto completo di codice a elevato accoppiamento e il Capitolo 3 mostra come reimplementare lo stesso esempio utilizzando la Dependency Injection. Rispetto alle altre parti, la Parte I ha una progressione più lineare in termini di contenuti. Per trarne il massimo è consigliabile leggere ogni capitolo dall'inizio.
La Parte II è un "catalogo" di pattern, anti-pattern e difetti di programmazione. È una guida prescrittiva sul modo in cui implementare la Dependency Injection e sui pericoli cui prestare attenzione. Il Capitolo 4 è un catalogo di pattern di progettazione della Dependency Injection e, viceversa, il Capitolo 5 è un catalogo di anti-pattern. Il Capitolo 6 contiene soluzioni generalizzate a problemi che si verificano comunemente. Come un catalogo, ogni capitolo contiene una serie di sezioni solo vagamente correlate e progettate per essere lette isolatamente e in sequenza.
La Parte III esamina la Dependency Injection da tre diverse angolazioni: *composizione dell'oggetto*, *gestione del ciclo di vita* e *intercettazione*. Nel Capitolo 7, discutiamo come implementare la Dependency Injection nei framework di applicazioni, ASP.NET Core e

UWP, e come implementare la Dependency Injection usando un'applicazione a console. Il Capitolo 8 descrive come gestire il ciclo di vita delle dipendenze, per evitare sprechi di risorse. Sebbene la struttura sia un po' meno rigorosa rispetto ai capitoli precedenti, gran parte del capitolo può essere utilizzata come catalogo di *stili di ciclo di vita* ben noti. I restanti tre capitoli spiegano come comporre applicazioni con *problematiche trasversali*. Il Capitolo 9 approfondisce le basi dell'*intercettazione* usando i decoratori, mentre i Capitoli 10 e 11 approfondiscono il concetto delle *Aspect Oriented Programming*. È qui che coglierete i benefici di tutto il lavoro svolto in precedenza, quindi, per molti versi, questo può essere considerato il culmine del libro.

La Parte IV è un catalogo di librerie di *container di Dependency Injection*. Inizia con una discussione su cosa sono i container di Dependency Injection e su come si collocano nel quadro generale. I restanti tre capitoli trattano ciascuno un container specifico in un certo dettaglio: Autofac, Simple Injector e Microsoft.Extensions.DependencyInjection. Ogni capitolo tratta un container in una forma piuttosto sintetica per risparmiare spazio, quindi potreste voler leggere solo quelli relativi a uno o due container che vi interessano di più. Per molti versi, consideriamo questi tre capitoli come delle "appendici".

Per fare in modo che la discussione dei principi e dei pattern di Dependency Injection si mantenga esente da qualsiasi API di container, la maggior parte del libro, a eccezione della Parte IV, è scritta senza far riferimento a un particolare container. Questo è anche il motivo per cui i container compaiono così pesantemente nella Parte IV. La nostra speranza è che, mantenendo generale la discussione, il libro possa risultare utile per un periodo di tempo più lungo.

Potete anche prendere i concetti dalle Parti da I a III e applicarli a librerie di container non trattate nella Parte IV. Sono disponibili buoni container che, sfortunatamente, non siamo riusciti a trattare. Ma speriamo che questo libro abbia molto da offrire anche agli utilizzatori di queste librerie.

Convenzioni e download del codice

Questo libro contiene numerosi esempi di codice. La maggior parte di questi è in linguaggio C#, ma c'è anche un po' di XML e JSON, qua e là. Per il codice sorgente nei listati e nel testo è stato usato un `carattere a larghezza fissa come questo`, per distinguerlo dal testo ordinario.

Tutto il codice sorgente del libro è scritto in C# e Visual Studio 2017. Le applicazioni ASP.NET Core sono scritte in ASP.NET Core v2.1.

Solo alcune delle tecniche descritte in questo libro dipendono da caratteristiche recenti del linguaggio. Intendevamo trovare un ragionevole equilibrio fra stili di programmazione tradizionali e moderni. Quando scriviamo codice professionale, utilizziamo in misura molto maggiore le funzionalità recenti del linguaggio, ma, per la maggior parte, le funzionalità più avanzate sono i generici e LINQ. L'ultima cosa che vogliamo è che vi facciate l'idea che la Dependency Injection possa essere applicabile solo impiegando linguaggi "ultramoderni".

Scrivere esempi di codice per un libro presenta tutta una serie di sfide. Rispetto a un monitor, un libro consente di usare solo righe di codice molto brevi. Sarebbe stato molto allettante scrivere il codice in uno stile conciso, usando nomi brevi ma criptici per i metodi e le variabili. Tale codice è già difficile da capire quando avete un IDE e

un debugger, ma diventa davvero ostico da seguire in un libro. Abbiamo pensato che era molto importante mantenere il più possibile la leggibilità dei nomi. Per "farci stare" tutto, a volte abbiamo dovuto ricorrere ad alcune interruzioni di riga "poco ortodosse". Tutto il codice è compilabile, ma a volte la formattazione vi sembrerà un po' strana.
Il codice utilizza anche la parola chiave C# `var`. Nel codice professionale, in cui la larghezza della riga non ha i limitati dimensionali della pagina di un libro, spesso utilizziamo uno stile di programmazione diverso quando applichiamo `var`. Qui, per risparmiare spazio, usiamo `var` ogni volta che riteniamo che una dichiarazione esplicita renda il codice meno leggibile.
Useremo spesso la parola *classe* come sinonimo di "tipo". In .NET le classi, le `struct`, le interfacce, le `enum` e così via sono tutti tipi, ma poiché la parola *tipo* è ricchissima di significati nel linguaggio comune, spesso un suo uso eccessivo minerebbe la chiarezza del testo.
La maggior parte del codice presente in questo libro si riferisce a un esempio generale che attraversa un po' tutte le pagine: un negozio online completo di applicazioni di gestione interne di supporto. Questo è l'esempio meno entusiasmante che potete aspettarvi di vedere in qualsiasi testo software, ma l'abbiamo scelto per due motivi.

- È un dominio di problemi ben noto alla maggior parte dei lettori. Anche se può sembrare noioso, pensiamo che questo sia un vantaggio, perché non distoglie l'attenzione dalla Dependency Injection.
- Dobbiamo anche ammettere che non siamo davvero riusciti a immaginare un altro dominio abbastanza ricco da supportare tutti i diversi scenari che avevamo in mente.

Abbiamo scritto molto codice per supportare gli esempi, e la maggior parte di quel codice non è riportata in questo libro. In effetti, abbiamo scritto quasi tutto utilizzando lo sviluppo TDD (*Test-Driven Development*), ma poiché questo non è un libro sul TDD, generalmente nel libro non abbiamo mostrato i test delle unità.
Il codice sorgente di tutti gli esempi del libro è disponibile sul sito web di Manning: `www.manning.com/books/dependency-injection-principles-practices-patterns` e sul sito web di Apogeo all'indirizzo `https://bit.ly/apo-depin`. Il file `README.md` nella root di download contiene le istruzioni per la compilazione e l'esecuzione del codice.

Gli Autori

Steven van Deursen è uno sviluppatore e un esperto di architettura .NET freelance olandese con esperienza nel settore dal 2002. Vive a Nijmegen e ama scrivere codice per divertimento e profitto. Oltre a scrivere codice, Steven pratica le arti marziali, ama uscire per mangiare e sicuramente ha sempre voglia di un buon whisky.
Mark Seemann è un programmatore, architetto software e relatore di Copenaghen. Maneggia il software dal 1995 e il TDD dal 2003, inclusi sei anni in Microsoft come consulente, sviluppatore e architetto. Mark è attualmente impegnato professionalmente nello sviluppo di software e lavora a Copenaghen. Gli piace leggere, dipingere, suonare la chitarra, il buon vino e il cibo gourmet.