Introduction

Visual C# .NET (C#) is relatively easy to learn for anyone familiar with another object-oriented language. Even someone familiar with Visual Basic 6.0, who is looking for an object-oriented language, will find C# easy to pick up. However, though C#, coupled with the .NET Framework, provides a quick path for creating simple applications, you still must know a wealth of information and understand how to use it correctly in order to produce sophisticated, robust, fault-tolerant C# applications. I teach you what you need to know and explain how best to use your knowledge so that you can quickly develop true C# expertise.

Idioms and design patterns are invaluable for developing and applying expertise, and I show you how to use many of them to create applications that are efficient, robust, fault-tolerant, and exception-safe. Although many are familiar to C++ and Java programmers, some are unique to .NET and its Common Language Runtime (CLR). I show you how to apply these indispensable idioms and design techniques to seamlessly integrate your C# applications with the .NET runtime, focusing on the new capabilities of C# 2.0.

Design patterns document best practices in application design that many different programmers have discovered and rediscovered over time. In fact, the .NET Framework itself implements many well known design patterns. Similarly, over the past three versions of the .NET Framework and the past two versions of C#, many new idioms and best practices have come to light. You will see these practices detailed throughout this book. Also, it is important to note that the invaluable tool chest of techniques is evolving constantly.

.NET 2.0 provides a unique and stable cross-platform execution environment. C# is only one of the languages that targets this powerful runtime. You will find that many of the techniques explored in this book are also applicable to any language that targets the .NET runtime.

For those of you who have significant C++ experience and are familiar with such concepts as C++ canonical forms, exception safety, Resource Acquisition Is Initialization (RAII), and const correctness, this book explains how to apply these concepts in C#. If you're a Java or Visual Basic programmer who has spent years developing your toolbox of techniques and you want to know how to apply them effectively in C#, you'll find out how to here.

As you'll see, it doesn't take years of trial-and-error experience to become a C# expert. You simply need to learn the right things and the right ways to use them. That's why I wrote this book for you.

About This Book

I assume that you already have a working knowledge of some object-oriented programming language, such as C++, Java, or Visual Basic (.NET or 2005). Since C# derives its syntax from both C++ and Java, I don't spend much time covering C# syntax, except where it differs starkly from C++ or Java. If you already know some C#, you may find yourself skimming or even skipping Chapters 1 through 3.

Chapter 1, "C# Preview," gives a quick glimpse of what a simple C# application looks like, and it describes some basic differences between the C# programming environment and the native C++ environment.

Chapter 2, "C# and the CLR," expands on Chapter 1 and quickly explores the managed environment within which C# applications run. I introduce you to assemblies, the basic building blocks of applications, into which C# code files are compiled. Additionally, you'll see how metadata makes assemblies self-describing.

Chapter 3, "C# Syntax Overview," surveys the C# language syntax. I introduce you to the two fundamental kinds of types within the CLR: value types and reference types. I also describe namespaces and how you can use them to logically partition types and functionality within your applications.

Chapters 4 through 13 provide in-depth descriptions on how to employ useful idioms, design patterns, and best practices in your C# programs and designs. I've tried hard to put these chapters in logical order, but occasionally one chapter may reference a technique or topic covered in a later chapter. It is nearly impossible to avoid this situation, but I tried to minimize it as much as possible.

Chapter 4, "Classes, Structs, and Objects," provides details about defining types in C#. You'll learn more about value types and reference types in the CLR. I also touch upon the native support for interfaces within the CLR and C#. You'll see how type inheritance works in C#, as well as how every object derives from the `System.Object` type. This chapter also contains a wealth of information regarding the managed environment and what you must know in order to define types that are useful in it. I introduce many of these topics in this chapter and discuss them in much more detail in later chapters.

Chapter 5, "Interfaces and Contracts," details interfaces and the role they play in the C# language. Interfaces provide a functionality contract that types may choose to implement. You'll learn the various ways that a type may implement an interface, as well as how the runtime chooses which methods to call when an interface method is called.

Chapter 6, "Overloading Operators," details how you may provide custom functionality for the built-in operators of the C# language when applied to your own defined types. You'll see how to overload operators responsibly, since not all managed languages that compile code for the CLR are able to use overloaded operators.

Chapter 7, "Exception Handling and Exception Safety," shows you the exception-handling capabilities of the C# language and the CLR. Although the syntax is similar to that of C++, creating exception-safe and exception-neutral code is tricky—even more tricky than creating exception-safe code in native C++. You'll see that creating fault-tolerant, exception-safe code doesn't require the use of `try`, `catch`, or `finally` constructs at all. I also describe some of the new capabilities within the .NET 2.0 runtime that allow you to create more fault-tolerant code than was possible in .NET 1.1.

Chapter 8, "Working with Strings," describes how strings are a first-class type in the CLR and how to use them effectively in C#. A large portion of the chapter covers the string-formatting capabilities of various types in the .NET Framework and how to make your defined types behave similarly by implementing `IFormattable`. Additionally, I introduce you to the globalization capabilities of the framework and how to create custom `CultureInfo` for cultures and regions that the .NET Framework doesn't already know about.

Chapter 9, "Arrays, Collection Types, and Iterators," covers the various array and collection types available in C#. You can create two types of multidimensional arrays, as well as your own collection types while utilizing collection-utility classes. You'll see how to define forward, reverse, and bidirectional iterators using the new iterator syntax introduced in C# 2.0, so that your collection types will work well with `foreach` statements.

Chapter 10, "Delegates, Anonymous Functions, and Events," shows you the mechanisms used within C# to provide callbacks. Historically, all viable frameworks have always provided a mechanism to implement callbacks. C# goes one step further and encapsulates callbacks into callable objects called *delegates*. Additionally, C# 2.0 allows you to create delegates with an abbreviated syntax called *anonymous functions*. Anonymous functions are similar to lambda functions in functional programming. Also, you'll see how the framework builds upon delegates to provide a publish/subscribe event notification mechanism, allowing your design to decouple the source of the event from the consumer of the event.

Chapter 11, "Generics," introduces you to probably the most exciting feature added to C# 2.0 and the CLR. Those familiar with C++ templates will find generics somewhat familiar, though many fundamental differences exist. Using generics, you can provide a shell of functionality within which to define more specific types at run time. Generics are most useful with collection types and provide great efficiency compared to the collections of previous .NET versions.

Chapter 12, "Threading in C#," covers the tasks required in creating multithreaded applications in the C# managed virtual execution environment. If you're familiar with threading in the native Win32 environment, you'll notice the significant differences. Moreover, the managed environment provides much more infrastructure for making the job easier. You'll see how delegates, through use of the I Owe You (IOU) pattern, provide an excellent gateway into the process thread pool. Arguably, synchronization is the most important concept when getting multiple threads to run concurrently. This chapter covers the various synchronization facilities available to your applications.

Chapter 13, "In Search of C# Canonical Forms," is a dissertation on the best design practices for defining new types and how to make them so you can use them naturally and so consumers won't abuse them inadvertently. I touch upon some of these topics in other chapters, but I discuss them in detail in this chapter. This chapter concludes with a checklist of items to consider when defining new types using C#.