Chapter 5

An Introduction to C#

5.1 Introduction

C# is a new programming language. There have only been a few programming languages introduced in the last 10 years that have had the impact that C# has had: Java, another object-oriented language, is the obvious example. In this chapter I will try to provide some background on the C# language, what the motivations behind C# are and how it fits into the grand scheme of things (from Microsoft's point of view as well as with respect to existing languages).

5.2 Background

First we need to (briefly) consider the .NET environment. .NET is Microsoft's new environment for software development on Windows-based machines. Some have pushed it as an environment for developing Web-based services (which it certainly is), but it is more than that: it is a new way for Windows applications to be designed, built, deployed and run. Rather than just updating the existing tools and languages (and so making them even more complex), Microsoft has started from the ground up with .NET. The aim was to create a development environment within which most programming tasks can be easily accomplished. A major component of .NET is the C# programming language.

C# (pronounced C-Sharp, just like in musical notation) is a new language for software development on Windows-based machines. It is heavily integrated with the .NET environment and is intended as an alternative to the main previous languages, Visual C++ and VB. It builds very heavily on Visual C++ and (let's be honest) Java. It is, however, its own language, and whilst borrowing from both adds new features (and again let's be honest: nothing is new under the Sun!).

Of course, one thing that has counted against Microsoft in the past has been its tendency to try to make everything it produces proprietary, and to restrict access to the inner details of its world. Possibly to counter this Microsoft has put the key .NET technologies forward as a standard. As part of this, C# was submitted for standardization to ECMA in late 2000. The ECMA (http://www.ecma.ch/), founded in 1961, is a vendor-neutral international standards organization committed to driving industry-wide adoption of information and communications technologies. The idea behind this (at least from the standards organization's point of view) is that this should make it possible for anybody who wishes to implement C# programming tools on any platform to do so. Microsoft has also submitted a subset of the Microsoft .NET Framework, called the Common Language Infrastructure (CLI), to ECMA. This could also make it possible for other vendors to implement the CLI on a variety of platforms (whether this actually happens or not we shall have to wait and see). The C# Language Specification and the Common Language Infrastructure were approved by the ECMA General Assembly on 13 December 2001 (see `http://www.ecma.ch/ecma1/STAND/ecma-334.htm` and `http://www.ecma.ch/ecma1/STAND/ecma-335.htm`.

An interesting point to note is that when C# and the Common Language Infrastructure were submitted they were actually submitted by Hewlett-Packard and Intel Corporation as well as by Microsoft.

5.3 What Is C#?

C# can be viewed from a number of perspectives; in this it differs from many other programming languages, which can only be viewed as a programming language and nothing else. However, C# is more than just a programming language. Below we consider some of the ways to classify C#:

- *An object-oriented programming language* C# certainly provides the syntax and semantics of an object-oriented language. It is supported by a compiler that takes programs written in C# and produces code that can be executed by the .NET runtime environment (more on this later). As for the C# language itself, it is rather compact, unlike languages such as Ada, which are very large.
- *A programming environment* I refer here to the presence of the system-provided classes (often referred to as the .NET Classes) rather than any particular development environment. Unlike many languages (including traditional C++), C# has associated with it a large (and fairly standard) set of classes. These classes make C# very powerful and promote a standard development style. You spend most of your time extending the "system" rather than programming from scratch. In a number of cases, these classes provide facilities that are considered part of the language in Ada, C and Pascal. The result is that C# is anything but a small programming system.
- *An operating environment* The operating environment is the Common Language Runtime (CLR) in which all C# programs execute (actually it goes wider than C# and incorporates the other .NET languages as well). This is C#'s run-time environment, which handles memory allocation and deallocation, references, flow of control, interaction with other .NET elements etc.
- *The language of Web Services* C# (and .NET) have received huge hype as the language which will bring the .NET framework and the associated Web Services alive. However, in many ways C# is just an object-oriented language. There is no particular reason why C# should be any better as a Web language than Smalltalk or any other interpreted object-oriented language, such as Objective-C or Eiffel. Indeed, Visual Basic or Visual C++ could just as easily be made

the language to leverage the .NET framework (and indeed Microsoft has extended each to work with the .NET framework, creating VB.NET and Managed C++). However, *C# has been designed specifically with the .NET framework in mind, and hence is very well structured for writing code that will be compiled for .NET.* It is also a green field language, in that it did not need to take into account any previous incarnations that might lead to the incorporation of legacy features in the language. It could also learn a few tricks from Java and C++ which it could use to its advantage (see later).

Thus it is quite possible to say that C# is a programming language, a set of extensible classes, an operating environment or even a Web development tool. It is, in fact, all of these.

5.4 Objects in C#

Almost everything in C# is an object; for example, strings, arrays, windows and even exceptions can be objects. Objects, in turn, are examples of classes of things; for example, the string "John Hunt" is an object of the class `String`. Thus, to program in C#, you define classes, create instances and apply operations to classes and objects.

However, unlike languages such as Smalltalk, which are considered pure object-oriented languages, C# also has standard types (such as `int`s, `float`s and `bool`s) and procedural programming statements. This hybrid approach can make the transition to object orientation simpler. In some languages, such as C++, this can be a disadvantage, as the developer can avoid the object-oriented nature of the language. However, C# has no concept of a procedural program; instead, everything is held within an object (even the procedural elements).

5.5 Commercial Versions of C#

At present the only vendor of a C# compiler is Microsoft itself; however, there is no reason why this should be the case. Indeed, there is already a project to take C# and port it to Linux. This is very exciting and would mean that C# programs would have the potential to be cross-platform. In theory, there is no reason why this should not be the case (C# compiles to an intermediate language which, like Java, could be interpreted by run-time environments ported to different platforms). However, in practice, because C# can exploit the whole of the .NET framework and the .NET classes, this would mean that the .NET framework would need to be ported to a new operating system (hardly something that Microsoft will be pushing!). The result is that it is likely that C# will make the move, but I suspect .NET will not (at least not in any major way).

5.6 The C# Environment

C# is different from other previous Microsoft languages that you may have used in that, when you write C# code, it does not execute on your host machine, even when it is compiled. Instead, it

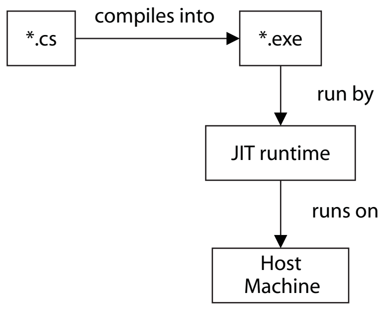

Figure 5.1 C# run-time environment.

executes in a virtual machine, which in turn executes on your host computer. This is because your C# code is compiled into an Intermediate Language (IL). This IL code is then run using a JIT run-time. A JIT is a Just-In-Time compiler that converts the IL code into an executable form the first time that code is encountered. All subsequent calls to that code can then use the dynamically generated native code.

This can look a little confusing at first sight, as the C# compiler (csc) appears to generate a `.exe` file. However, if you refer to these executables as "managed executables" that will be "managed" by your run-time environment then you are not far off the mark.

As an example of what is actually going on, take a look at Figure 5.1. In this figure we compile our C# program into the IL form using the C# compiler (called csc). When we execute this a JIT run-time system is used to run it on the host machine.

5.7 Comparing C# to Java and C++

Although if you read any of the material from Microsoft you will not find any reference to Java there, C# owes a very big debt to both Java and C++. Many of the ideas in C# can be found in these languages. Indeed, C# uses the same syntax as Java for comments, flow of control commands and exception handling, while 46 of its 50 keywords and 47 of its 56 operators are similar or identical to their Java counterparts. However, this does not mean that C# is Java or indeed that it is just an extension to C++. As an exercise of interest to the reader, the similarities with C++ and Java are listed in Table 5.1. Note that in the table, C++ relates to the original C++ language and not to the extensions provided by Microsoft for C++ in the .NET framework.

As stated earlier C# is not Java and it is not C++; indeed, C# introduces a number of new concepts itself, summarized in Table 5.2.

Each of the features described in this section relating to C# will be explained in more detail later in this book. However, what you should take from this is that C# has things in common with both C++ and Java, as well as introducing some new features.

Table 5.1 Comparison of features in Java, C# and C++.

Feature	*Java*	*C#*	*C++*
Object-oriented programming language	Yes	Yes	Yes
Single inheritance	Yes	Yes	No
Single class hierarchy root	Yes	Yes	No
Automatic memory management	Yes	Yes	No
Reusable components and information on how to use and deploy	Yes	Yes	No
A large library of base components	Yes	Yes	No
Web page development framework	Yes	Yes	No
Database access framework	Yes	Yes	No
Middleware and application integration	Yes	Yes	No
Inner/nested classes	Yes	Yes	No
Interfaces	Yes	Yes	No
Templates	No	No	Yes
Goto statement	No	Yes	Yes
Enumerated types	No	Yes	Yes
Operator overloading	No	Yes	Yes
Destructors	No	Yes	Yes
Compiler directives	No	Yes	Yes
Header files	No	No	Yes

Table 5.2 New features in C#.

Features	*Description*
Unified type system	Primitive data types inherit from parent object, dispensing with need for wrapper classes like Int for integers or Double for doubles.
Rich parameter-passing syntax	`in`, `out`, `ref` and `params` allow for easier interfacing with other languages and systems.
Delegate functions	References to functions are. made type-safe and secure
Structs	Similar to classes, except that they cannot inherit and they are value types. Structs are used to optimize code performance.
Properties	Simplify the syntax of getting and setting single-field values in a class.
Indexers	Simplify the syntax for getting at an array of object values.
Attributes	Assign run-time values based on compilation steps.

5.8 C# Keywords

Table 5.3 presents all the C# keywords. Do not worry about what they mean too much at the moment; we will be covering all of them during the course of this book.

Table 5.3 C# keywords.

`abstract`	`base`	`bool`	`break`	`byte`
`case`	`catch`	`char`	`checked`	`class`
`const`	`continue`	`decimal`	`default`	`delegate`
`do`	`double`	`else`	`enum`	`event`
`explicit`	`extern`	`false`	`finally`	`fixed`
`float`	`for`	`foreach`	`goto`	`if`
`implicit`	`in`	`int`	`interface`	`internal`
`is`	`lock`	`long`	`namespace`	`new`
`null`	`object`	`operator`	`out`	`override`
`params`	`private`	`protected`	`public`	`readonly`
`ref`	`return`	`sbyte`	`sealed`	`short`
`sizeof`	`static`	`string`	`struct`	`switch`
`this`	`throw`	`true`	`try`	`typeof`
`uint`	`ulong`	`unchecked`	`unsafe`	`ushort`
`using`	`virtual`	`void`	`while`	

5.9 Where to Get More Information

There are numerous places on the Web that can provide useful information for the C# developer. Some of these are:

- `http://msdn.microsoft.com/vstudio/technical/articles/Csharpintro.asp`
- `http://www.microsoft.com/net/default.asp`: .NET Platform home site
- `http://www.csharphelp.com/`: CSharpHelp.Com
- `http://www.c-sharpcorner.com/`: C# Corner
- `http://www.csharptoday.com/`: CSharpToday.Com
- `http://www.codehound.com/csharp/`: Code Hound C# Search Engine
- `http://www.csharp-station.com/`: C# Station
- `http://www.cshrp.net/`: CShrp.Net
- `http://www.csharpindex.com/`: CSharpIndex.Com