C# Concisely

Judith Bishop
Nigel Horspool

Judith Bishop, University of Pretoria, South Africa   jbishop@cs.up.ac.za
Nigel Horspool, University of Victoria, Canada   nigelh@uvic.ca

April 2003

Addison-Wesley Publishers, Harlow, UK

April 7, 2003 11:01 am Production-TOC-etc.fm

Contents

Preface

C# Concisely has been a year in the making. We'd like to tell you, the reader, instructor or student, why we embarked on this project, and what is in store for you in this book.

1 C# – where from and where to

C# is a new language designed by Anders Hejlsberg, Scott Wiltamuth and Peter Golde at Microsoft to be the main development medium for the .NET Framework and all future Microsoft products. C# has its origins in other languages, chiefly C++, Java, Modula-2 and Smalltalk. Hejlsberg's credentials include being chief architect behind Turbo Pascal and Borland's Delphi, and the depth of experience shows through in a very nicely designed language. More so than the languages mentioned above, C# features deep object-orientation and a new concept – object-simplification – which makes it much easier to enter the world of object oriented programming, and stay there.

At this stage, C# is manœuvering its place in the languages chosen for software development and teaching. Since *C# Concisely* is chiefly a textbook, we can say that it is our opinion that C# would be an excellent first teaching language. The main barrier against it reaching that position is inertia: Java is firmly entrenched in most institutions, and a great deal of effort and infrastructure supports its position.

C# is also a good language for later courses. Because it supports more advanced programming features such as delegates and operator overloading, and there is even a proposal out for generics, it could be an excellent vehicle for courses such as Advanced Programming, Data Structures, Net-centric Computing and Distributed Systems, among others. It is also likely that postgraduate courses with an industrial bias, such as many course-based MSc programs, will benefit from C# since it equips their students for the workplace well.

Unlike Java, C# is only free to use in certain circumstances. If an academic institution belongs to the Microsoft Academic Alliance (available at minimal cost), then the software is available for all staff and students. The C# language (but not all of its libraries) has also been ratified as a standard by the European body ECMA (December 2001), and as a result, other compilers for Windows and other platforms are now available, and these are also free. Microsoft itself has produced three of these under the code name Rotor for the Windows, FreeBSD (April 2002) and Macintosh OSX (November 2002) platforms.

A concern with Microsoft products in general is that they tend to be resource hungry, and require computers beyond the means of many student laboratories. The free compilers are therefore an attractive option. However, a notable omission from the ECMA standard was the `Windows.Form` API which is used for programming graphical user interfaces. We have worked with Microsoft since April 2002 to fill this gap, and the result is Views, a small-footprint XML-based GUI system. Versions of Views exist for both the standard Windows system and the other platforms which host Rotor. Together, Views and freely available C# compilers offer a viable, and technologically modern, alternative to a purchased C# and Visual Studio combination from Microsoft, with the added advantage, of course, of being platform independent.

C# definitely has a future in teaching and research, as well as in system development. The objective of this book is to promote that future by spreading the use of the language as widely as possible.

2 Outline of the book

It has become almost a cliche these days that a programming textbook describes itself as having an “objects first” approach. When digging deeper into text books, and into papers presented at computer science educational conferences, it became clear to us that the meaning of the term “objects first” was pretty vague. Moreover, it was seldom indicated what came second or third.

Under any definition, *C# Concisely* is an objects-first book, but we believe we have crystallised the meaning of the term with a novel approach. In Chapter 2 we introduce objects first by *using* them rather than defining types for them. We rely on two built-in structured types, `DateTime` (a struct) and `Random` (a class), and develop the concepts related to instantiation, variables, methods and assignment in a much more natural way than would be possible if the types had to be defined first.

Then in Chapter 3, we look at how to define a type, going back and consolidating concepts with which the reader is familiar, and adding more formality to terms like parameters and properties. We have taken the decision to remain firmly in the value-world at this stage, using structs as the means to create objects, and mentioning references only for parameter passing. We believe that covering one concept at a time is good pedagogy, and that the confusion often associated with value *vs*. reference types can be avoided by delaying the detailed treatment of references until Chapter 7. The result of this decision is that classes are also only covered in Chapter 7. More so than other current object-oriented languages, C# emphasises the concept of a type, and it is treated the same whether realised as an int, a struct or a class. We adopt the same view, and therefore delaying classes till later does not cause unnatural programming.

The question of “if objects are first, what is second?” is answered in Chapter 4: control structures, strings and arrays. These are all essential building blocks for writing meaningful programs. Without them, the more powerful features of object-orientation – collections and polymorphism – cannot be exercised properly.

Before we get there, though, we spend two useful chapters on laying a foundation for meaningful programming. Chapter 5 describes the Views system which we have written as an adjunct to any C# program for describing and handling graphical user interfaces. With such a tool, programs that interact with the user can be developed rapidly and in a platform independent manner.

Chapter 6 is almost unique in programming textbooks in that it tackles the issue of errors and debugging. A range of common and novel debugging techniques from print statements to assert calls is described, most of which require no additional software other than a C# compiler. There is a separate section on using a debugger, and more information on Microsoft’s Visual Studio capabilities in an appendix.

At this point, programmers will be able to handle any task that would have been possible in an older language, but in an object-oriented way. Chapters 7 and 8 complete the coverage of objects by dealing with collections (which C# does particularly well) and extensibility and polymorphism. In Chapter 8, we take care to talk about principles, and then show how they can be realised in any of the three techniques provided – inheritance, interfaces and abstract classes.

The intention for Chapters 9 and 10 is to cover network interaction, threads, delegates and an example of web services. It is envisaged that these two chapters would be optional in a first course.

3 Approach

C# Concisely teaches by means of an even mixture of formality and examples. Each new construct or built-in type (API) is described in a “form” which describes its syntax and semantics, and then small examples are followed by completely worked out programs. There are over 50 such programs in the book, some of which progress from chapter to chapter, showing how new features can produce more complete solutions to the original problem.

GUIs are important, as is testing. Considerable reliance is placed on the random generator for producing test data where possible, so that programs can be properly exercised and have meaningful amounts of output data.

No language presents a neat sequence of features, each of which can be covered in isolation, and C# is no exception. We have adopted a spiral approach for several features such as output, strings, loops and arrays, introducing them in a simple way early on, and then completing their description later.

A good textbook should be as interactive as possible, and we have provided several learning aids, both in the book itself and on the accompanying website (planned). These include:

- extensive appendices listing key APIs and grouping features of the language for easy reference
- ten-point multiple choice questions after every chapter
- plenty of exercises of different levels of difficulty after every chapter
- all the examples on-line at http://www.cs.up.ac.za/csharp
- the complete Views system, with Reference Manual and examples
- a CD containing all the above
- a discussion board for questions about C#, the book and Views.
- slides and answers for the instructor on a separate website.

4 Views

The Views system was developed by the authors as a project under the Rotor initiative started by Microsoft in April 2002. It has been class tested with first and second year students, who were enthusiastic about it as an alternative to programming with the Windows.Forms API or Visual Studio. Views is currently (January 2003) at version 1.16 and will continue to be enhanced and updated until the project ends in September. Further updates can be obtained from the website at `http://www.cs.up.ac.za/rotor`.

5 Acknowledgements

We would like to acknowledge the considerable practical input to this book and to Views made by students in the Polelo Group at the University of Pretoria – Basil Worrall, Kathrin Berg, Johnny Lo, David-John Miller, Mandy van Schalkwyk and Sfiso Shabalala. Many thanks to Edwin Peer who developed and maintained the web-based repository system which supported the our book so well. At the University of Victoria, Rajwinder Panesar-Walawege was responsible for the implementation of Views for Rotor, and Bernhard Scholz (on leave from the Technical University of Vienna) worked on an essential front end to Views, to be released later.

We would like to thank Microsoft for their generous gift which supported this book and Views, and JMB is particularly appreciative of the initiative and support given by Leanne Scott-Williams of Microsoft (South Africa). This work was also supported by a matching THRIP grant from the South African National Research Foundation. NH adds thanks to Pamela Lauz of Microsoft Canada for her support.

Working with a ten hour timezone difference, we could not have survived without the excellent internet support provided by our departments at the Universities of Pretoria and Victoria. Since much of the work on the book took place at strange hours of the day, NH would like to thank Torrefazione Italia for serving much needed and delicious coffee and KBSG Seattle for supplying the right kind of background music. JB acknowledges gratefully the contribution of the Cape wine industry, and the Late Late Show on Classic FM.

Finally, our thanks go to our families for their forbearance during the long hours spent at home working on this project, and to the stimulating discussions which they, as computer-oriented as we are, enabled us to have. We hope they will be as pleased with the results as we are.