# Preface

## Second Edition

I started writing the first edition of this book in 1988, and it was finally published in the last days of 1990. The eight years between the original development of the book and the present have seen a series of changes in the field of object-oriented programming, necessitating almost a complete rewriting of the text. Among these changes are the following:

- A better understanding of the distinction between subclasses and subtypes, and an appreciation of the fact that the two are often not the same.
- The rapid growth, evolution and standardization of the language C++, including the introduction of templates, exceptions, name spaces, booleans, strings, the RTTI (run-time type identification system), and the standard library.
- The introduction of the programming language Java, as an exciting new tool for developing applications for the World-Wide-Web.
- The slow demise of the language Object Pascal after being abandoned by Apple as the primary language for application development on the Macintosh, only to be reborn as a new language with the same name as a tool for developing PC applications using Delphi.
- The equally slow collapse in the use of the language Objective-C. For those interested in the study of languages this is an unfortunate loss, since the dynamically typed Objective-C is almost an ideal foil to the statically typed language C++. For this reason I have continued to discuss Objective-C in the second edition of the book.
- The development of exciting new object-oriented languages, such as Beta, CLOS and Java, that bring both new ideas and old ideas wrapped up in new guises.
- The development of new ways of thinking about *collections* of classes that work together, resulting in the refinement of ideas such as application frameworks and design patterns.

For these reasons and many more almost every section of the second edition has been revised from the first. Nevertheless, I have attempted to retain the overall structure of the book. This structure can be described as a series of themes, somewhat as follows:

**I. Introduction, design**. Chapter 1 introduces in an informal setting the basic concepts of object-oriented programming. Chapter 2 continues this theme, by introducing the principle of designing by responsibility. These two chapters are fundamental, and their study should not be given short shrift. In particular, I strongly encourage at least one, if not several, group exercises in which CRC cards, introduced in Chapter 2, are used as a vehicle in problem solving. The manipulation of physical index cards in a group setting is one of the best techniques I have encountered for developing the notions of behavior, responsibility and encapsulation.

**II. Classes, Methods and Messages**. Chapters 3 and 4 introduce the basic syntax used by our example languages (Smalltalk, C++, Java, Objective-C and Object Pascal) to create classes and methods, and to send messages. Chapter 3 concentrates on the static features (classes and methods), while Chapter 4 describes the dynamic aspects (creating objects and sending messages). Chapters 5 and 6 then reinforce these ideas with the first of a series of *case studies*; example programs developed in an object-oriented fashion and illustrating various features of the technique.

**III. Inheritance and software reuse**. Chapters 7, 8 and 9 introduce the concept of inheritance, and the use of inheritance as a technique for software reuse. The case study in Chapter 8, written in the language Java, also illustrates the use of a standard API (Application Programming Interface). Chapter 9 contrasts the use of inheritance and composition as dual techniques for software reuse.

**IV. Inheritance in more detail**. The four chapters beginning with Chapter 10 delve into the concept of inheritance in greater detail. The introduction of inheritance into a programming language has an impact on almost every other aspect of the language, and these implications are often not initially obvious to the student (or programmer). Chapter 10 discusses message lookup and the binding of methods to messages, illustrating the fact that subclasses and subtypes need not be the same. Chapter 11 discusses the semantics of method overriding, pointing out two very different interpretations that are found in almost all languages. Chapter 12 continues by exploring some of the further implications of inheritance on topics such as memory allocation, assignment, and the testing for equality. Finally, Chapter 13 explores the idea of multiple inheritance.

**V. Polymorphism**. Much of the power of object-oriented programming comes through the application of various forms on polymorphism. Chapter 14 introduces the basic mechanisms used for attaining polymorphism in object oriented languages. This is followed by a pair of case studies that illustrate these points. The first, in Chapter 15, considers the creation of general purpose container libraries. A specific library, the recently developed Standard Template Library in C++, is then discussed in Chapter 16.

**VI. Software Engineering**. Chapter 17 discusses a number of standard software engineering topics in the context of object-oriented languages. Chapter 18 introduces some relatively new concepts in the field of object-oriented design, the ideas of *application frame-*

*works* and *design patterns*. Both are techniques for describing how collections of classes are used together in the solution of programming problems. Finally, Chapter 19 illustrates the design of an application framework by describing a specific example, the Little Application Framework.

**VII. Advanced Topics**. The concept of classes is, on second examination, not as simple as the development in Chapter 3 would lead us to believe. Chapter 20 briefly considers a number of advanced topics in the field of object-oriented programming. Examples include the idea of delegation (object-oriented programming without classes), and metaclass programming (programming the language itself). Chapter 21 describes in general terms the variety of implementation techniques used in the execution of object-oriented languages.

In the ten-week course I teach at Oregon State University I devote approximately one week to each of the major areas described above. Students are at the same time working on moderate-sized projects using an object-oriented language of their choice, and the term ends with student presentations of their project designs and outcomes.

In the first edition of the book I concluded with a chapter title “further information”. Unfortunately, the field moved so quickly that this information was almost immediately out of date. Rather than placing this information in the book, for the second edition I will endeavor to maintain a web site with recent information.

### Obtaining the Source

Source code for the cast studies presented in the book can be accessed via anonymous ftp from the machine `ftp.cs.orst.edu`, in the directory `/pub/budd/oopintro`. This directory will also be used to maintain a number of other items, such as an errata list, study questions for each chapter, and copies of the overhead slides I use in my course. This information can also be accessed via the world-wide-web, starting from my personal home pages at `http://www.cs.orst.edu/~budd/oopintro`. Requests for further information can be forwarded to the electronic mail address `budd@cs.orst.edu`. A paper mail address is: Professor Timothy A. Budd, Department of Computer Science, Oregon State University, Corvallis, Oregon, 97331, USA.

## Necessary Background

I have presented the material in this book assuming only that the reader is knowledgeable in some conventional programming language, such as Pascal or C. In my courses, the material has been used successfully at the upper-division (junior or senior) undergraduate level and at the first-year graduate-student level. In some cases (particularly in the last quarter of the book), further knowledge may be helpful, but is not assumed. For example, a student who has taken a course in software engineering may find some of the material in Chapter 17 more relevant, and one who has a course in compiler construction will find Chapter 21 more intelligible. On the other hand, both chapters can be simplified in presentation if necessary.

# Preface to First Edition

The inspiration to write this book arose, as did that for my earlier book on Smalltalk [Budd 87], when I was faced with teaching a course and was not able to find a suitable existing text. I had been for some years teaching a seminar on Smalltalk and object-oriented programming, using my own book as the text. Starting in the late 1980's, I received an increasing number of requests for a course structured around C++. About the same time, the popularity of the Macintosh computer brought with it a slightly smaller call for instruction in Object Pascal. Finally, the announcement of the NeXT computer resulted in inquiries about how best to learn Objective-C.

Since I did not wish to teach four different courses, each dealing with a specific object-oriented language, I resolved to teach a single course in which I would lay out the principles of object-oriented programming, illustrating these principles with examples from each of the four languages. Participants would learn a little about each language, and would complete a project in a language of their choice.

I then set out to find a textbook for such a course. What I discovered, to my surprise, was that existing texts, although in many ways quite admirable, were all oriented around a single language. Books I considered included Cox [Cox 86], Goldberg and Robson [Goldberg 83], Kaehler and Patterson [Kaehler 86], Keene [Keene 89], Meyer [Meyer 88a], Pinson and Wiener [Pinson 88] and its companion, Wiener and Pinson [Wiener 88], Pohl [Pohl 89], and Stroustrup [Stroustrup 86]. Although in the end, I selected a few of these as optional texts, I rejected all of them for use as a primary text for the simple reason that each one, to a greater or lesser extent, gives the impression that "object-oriented programming" is synonymous with "object-oriented programming in **X**", where **X** is whatever programming language happens to be the author's favorite. Instead, I started writing my own lecture notes to use as a primary text. Over the course of the next year I revised and extended these notes; this book is the result.

Various participants in my course (which turned out to be much more popular, and hence much larger, than I had anticipated), in addition to projects in the four languages that I have mentioned, successfully completed projects in Actor [Actor 87], Turbo Pascal [Turbo 88], and CLOS [Keene 89]. Since my objective was to convey the principles of object-oriented programming in a language-independent manner, I inquired specifically of these individuals whether the material discussed in my lecture notes was applicable and useful in their individual work. Based on their positive response, I believe I have at least partially succeeded in achieving a measure of language independence.

This book should *not* be considered a substitute for either a language tutorial or a language reference manual for any of the four languages discussed. In each of the languages, there are numerous subtle but language-specific or implementation-specific features that I did not believe were relevant to the discussions in this text, but that are certainly important as practical matters to the programmer.

## Acknowledgments

I am certainly grateful to the 65 students in CS589 at Oregon State University who, in the fall of 1989, suffered through the development of the first draft of this text. They received one chapter at a time, often only a day or two before I would lecture on the material. Their patience in this regard is appreciated. Their specific comments, corrections, critiques and criticisms were most helpful. In particular, I wish to acknowledge the detailed comments provided by Thomas Amoth, Kim Drongesen, Frank Griswold, Rajeev Pandey, and Phil Ruder.

The solitaire game developed in Chapter 8 was inspired by the project completed by Kim Drongesen, and the billiards game in Chapter 6 was based on the project completed by Guenter Mamier and Dietrich Wettschereck. In both cases, however, the code itself has been entirely rewritten and is my own. In fact, in both cases my code is considerably stripped down, for the purposes of exposition, and is in now way comparable to the greatly superior projects completed by those students.

I am also grateful to those people who provided comments, corrections, critiques, and criticisms on subsequent drafts of the manuscript. These individuals include Michael Adar, Jerrie Andreas, Brad Cox, Graham Dumpleton, Peter Grogono, Nola Hague, Marcia Horton, Ralph Johnson, Doug Lea, Ted Lewis, Stanley Lippman, Darcy McCallum, Lindsay Marshall, Makku Sakkinen, Michael Share, Dave Taenzer, Nabil Zamel, several reviewers, including Ed Gehringer, James Heliotis, Karl Lieberherr, Jeff Parker, Justin Smith, and Daniel Sterms and Keith Wollman, my editor at Addison-Wesley.

The source listings were printed using Latex macros based on C program formatting macros originally written by Éamonn McManus, of Trinity College, Dublin.

For an author it is always useful to have others provide an independent perspective on ones work, and I admit to gaining some useful insights into the first edition from a study guide prepared by Ms. K Brintz, Ms L Leenen, Mr TA Meyer, Ms JA Rosenblatt and Mrs AP Viljoen of the Department of Computer Science and Information Systems in the University of South Africa, in Pretoria, South Africa.

Countless people have provided assistance by pointing out errors or omissions in the first edition, and describing improvements. I am grateful to them all, and sorry I cannot name more people here.

For the second edition my capable, competent, and patient editors at Addison-Wesley have been Lynne Doran Cote and Deborah Lafferty.