An Introduction to Object-Oriented Programming (3rd Ed)

Timothy A. Budd
Oregon State University
Corvallis, Oregon
USA

September 12, 2001

This page is not blank.

Preface

I started writing my first book, on Smalltalk, in 1983. I can distinctly remember thinking that I needed to write quickly, so as to not miss the crest of the Object-Oriented programming wave. Who would have thought that two decades later object-oriented programming would still be going strong. What a long strange trip its been.

In the two decades that object-oriented programming has been studied, it has become *the* dominant programming paradigm. In the process it has changed almost every facet of computer science. And yet I find that my goal for the third edition of this book has remained unchanged from the first. It is still my hope to impart to my students, and by extension to my readers, an understanding of object-oriented programming based on general principles, and not specific to any particular language.

Languages come and go in this field with dizzying rapidity. In the first edition I discussed Objective-C and Apple's version of Object Pascal, both at that time widely used. Although both languages still exist, neither can at present be considered a dominant language. (However, I continue to talk about Objective-C in the third edition, because from a language point of view it has many interesting and unique features). Between the first edition and the third many languages seem to have disappeared (such as Actor and Turing), others have come into existence (such as Java, Eiffel and Self), many existing languages have acquired object extensions (such as Common Lisp and Object Perl), and many have burst on to the scene for a short while, then disappeared (for example, Sather and Dylan). Then there is Beta, a language that hints at wonderful ideas behind an incomprehensible syntax. Prediction is difficult, particularly about the future. Will languages that are just now appearing, such as Ruby, have staying power or will they go the way of Dylan? What about C#? It is difficult to imagine that any language with Microsoft behind it will fail to be successful, but stranger things have happened. (Personally, I think that C# will last because it presents a route for Visual Basic programmers to finally progress to a better language, but that few Java or C++ programmers will migrate to the new language. Time will tell if my powers of foresight are better than those of anybody else.)

For the present edition I have expanded the number of languages that I use for examples, but have eliminated many long narratives on a single language. Descriptions of techniques are often given in the form of tables or shorter explanations. As with the first two editions, I make no pretenses of being a reference manual for any language; and the student producing anything more than trivial programs in any of the languages I discuss would do well to avail themselves of a language-specific source.

Nevertheless, in this the third edition I have attempted to retain the overall structure I used in the first two editions. This can be described as a series of themes, as follows:

I. Introduction and Design. Chapter 1 introduces in an informal setting the basic concepts of object-oriented programming. Chapter 2 continues this by discussing the tools used by computer scientists to deal with complexity, and how object-oriented techniques fit into this framework. Chapter 3 introduces the principle of designing by responsibility. These three 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 3, are used in problem solving. The manipulation of physical index cards in a group setting is one of the best techniques I have encountered for developing and reinforcing the notions of behavior, responsibility, and encapsulation.

In the past decade the field of object-oriented design has expanded considerably. And for many readers Chapter 3 may either be too little or too much. Too much if they already have extensive experience with object oriented modeling languages and design, and too little if they have never heard of these topics. Nevertheless, I have tried to strike a balance. I have continued to discuss responsibility-driven-design, although it is now only one of many alternative object-oriented design techniques, because I think it is the simplest approach for beginning students to understand.

II. Classes, Methods, and Messages. Chapters 4 and 5 introduce the basic syntax used by our example languages (Smalltalk, C++, Java, Objective-C, Object and Delphi Pascal, and several others) to create classes and methods and to send messages. Chapter 4 concentrates on the compile-time features (classes and methods), while Chapter 5 describes the dynamic aspects (creating objects and sending messages). Chapters 6 and 7 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. Although inheritance is introduced in Chapter 1, it does not play a prominent role again until Chapter 8. Inheritance and polymorphic substitution is discussed as a primary technique for software reuse. The case study in Chapter 9, written in the newly introduced language C#, both illustrates the application of inheritance, and also illustrates the use of a standard API (application programming interface).

IV. Inheritance in More Detail. Chapters 10 through 13 delve into the concepts of inheritance and substitution in greater detail. The introduction of inheritance into a programming language has an impact on almost every other aspect of the language, and this impact is often not initially obvious to the student (or programmer). Chapter 10 discusses the sometimes subtle distinction between *subclasses* and *subtypes*. Chapter 11 investigates how different languages approach the use of static and dynamic features. Chapter 12 examines some of the surprising implications that result from the introduction of inheritance and polymorphic substitution into a language. Chapter 13 discusses the often misunderstood topic of multiple inheritance.

V. Polymorphism. Much of the power of object-oriented programming comes through the application of various forms of polymorphism. Chapter 14 introduces the basic mechanisms used for attaining polymorphism in object-oriented languages, and is followed by four chapters that explore the principle forms of polymorphism in great detail.

VI. Applications of Polymorphism. Chapter 19 examines one of the most common applications of polymorphism, the development of classes for common data structure abstractions. Chapter 20 is a case study that examines a recent addition to the language C++, the STL. Chapter 21 presents the idea of *frameworks*, a popular and very successful approach to software reuse that builds on the mechanisms provided by polymorphism. Chapter 22 describes one well known framework, the Java Abstract Windowing Toolkit.

VII. Object Interactions. Starting in Chapter 23 we move up a level of abstraction, and consider classes in more general relationships, and not just the parent/child relationship. Chapter 23 discusses the ways in which two or more classes (or objects) can interact with each other. Many of these interactions have been captured and defined in a formalism called a *design pattern*. The concept of design patterns and a description of the most common design patterns is presented in Chapter 24.

VIII. Advanced Topics. The final three chapters discuss topics that can be considered advanced for an introductory text such as this one. These include the idea of reflection and introspection (Chapter 25), network programming (Chapter 26) and the implementation techniques used in the execution of object-oriented languages (Chapter 27).

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 in this course are upper division undergraduate and first year graduate students. In conjunction with the lectures students work on moderate-sized projects using an object-oriented language of their choice, and the term ends with student presentations of project designs and outcomes.

Any attempt to force a complex and multifaceted topic into a linear narrative will run into issues of ordering, and this book is no exception. In general my approach has been to introduce an idea as early as possible, and in later chapters explore the idea in more detail, bringing out aspects or issues that might not be obvious on first encounter. Despite my opinion that my ordering makes sense, I am aware that others may find it convenient to select a different approach. In particular, some instructors find it useful to bring forward some of the software engineering issues that I postpone until Chapter 23, thereby bringing them closer to the design chapter (Chapter 3). Similarly, while multiple inheritance is a form of inheritance, and therefore rightly belongs in Section IV, the features that make multiple inheritance difficult to work with derive from interactions with polymorphism, and hence might make more sense after students have had time to read Section V. For these reasons and many more instructors should feel free to adapt the material and the order of presentation to their own particular circumstance.

Assumed 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 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 23 more relevant, and one who has had a course in compiler construction will find Chapter 27 more intelligible. Both chapters can be simplified in presentation if necessary.

Many sections have been marked with an asterisk (*). These represent optional material. Such sections may be interesting, but are not central to the ideas being presented. Often they cover a topic that is relevant only to a particular object-oriented language, and not to object-oriented programming in general. This material can be included or omitted at the discretion of the instructor, depending upon the interests and backgrounds of the students and the instructor, or dictates of time.

Obtaining the Source

Source code for the case studies presented in the book can be accessed via the mechanism of 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, from my personal home pages at `http://www.cs.orst.edu/~budd`. Requests for further information can be forwarded to the electronic mail address `budd@cs.orst.edu`, or to Professor Timothy A. Budd, Department of Computer Science, Oregon State University, Corvallis, Oregon, 97331.

From the Preface to 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 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, Booleans, name spaces, strings, the RTTI (run-time type identification system), and the standard library.
- The introduction of the programming language Java, an exciting new tool for developing applications for the World Wide Web.
- The slow demise of the language Object Pascal after its abandonment 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 decline in use of the language Objective-C. For those interested in 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 this edition.
- The development of interesting new object-oriented languages, such as Beta, CLOS, and Java, that incorporate both new ideas and old ideas 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 this edition has been revised.

From the Preface to First Edition

The inspiration to write this book arose, as it did for my earlier book on Smalltalk [Budd 1987], when I was faced with teaching a course and was not able to find a suitable existing text. I had for some years taught a seminar on Smalltalk and object-oriented programming, using my own book as the text. Starting in the late 1980s, I received an increasing number of requests for a course structured around C++. At 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 1986], Goldberg and Robson [Goldberg 1983], Kaehler and Patterson [Kaehler 1986], Keene [Keene 1989], Meyer [Meyer 1988a], Pinson and Wiener [Pinson 1988] and its companion, Wiener and Pinson [Wiener 1988], Stroustrup [Stroustrup 1986], and Pohl [Pohl 1989]. Although in the end I selected a few of these as optional texts, I rejected all of them as a primary text for the simple reason that each, 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 completing projects in the four languages that I mentioned above, successfully completed projects in Actor [Actor 1987], Turbo Pascal [Turbo 1988], and CLOS [Keene 1989]. Since my objective was to convey the principles of object-oriented programming independent of a specific language, I inquired of these individuals whether the material discussed in my lecture notes was applicable and useful in their work. On the basis of 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 my course, CS589, at Oregon State University who, in the fall of 1989, suffered through the development of the first draft of the first edition of this text. They received one chapter at a time, often only a day or two before I lectured 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 9 was inspired by the project completed by Kim Drongesen, and the billiards game in Chapter 7 was based on the project 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 no way comparable to the greatly superior projects completed by those students.

For an author it is always useful to have others provide an independent perspective on ones work, and I admit to gaining useful insights into the first edition from a study guide prepared by Arina Brintz, Louise Leenen, Tommie Meyer, Helene Rosenblatt, and Anel Viljoen of the Department of Computer Science and Information Systems at the University of South Africa, in Pretoria.

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

I benefitted greatly from comments provided by several readers of an early manuscript draft of this third edition. These reviewers included Ali Behforooz (Towson University), Hang Lau (Concordia University, Canada), Blayne Mayfield (Oklahoma State University), Robert Morse (University of Evansville), Roberto Ordóñez (Andrews University), Shon Vick (University of Maryland, Baltimore County), and Conrad Weisert (Information Disciplines, Inc.). I have made extensive revisions in response to their comments, and therefore any remaining errors are mine alone, and no reflection on their efforts.

For the third edition my capable, competent, and patient editor at Addison-Wesley has been Susan Hartman-Sullivan, assisted by Elinor Actipis. Layout and production was created by Paul Anagnostopoulos of Windfall, Inc. I have worked with Paul on several books now, and I'm continually amazed with the results he is able to achieve from my meager words.

Contents