Web Site

We are posting materials relevant to this book at www.modelsoftcorp.com.

Acknowledgements

Over the years we have worked with a number of people who have taught us about their business, given us the opportunity to try our ideas, and helped us learn in the process. We would like to thank these many individuals, starting with our former management at the General Electric R&D Center and continuing to our recent business friends and clients.

Although only two of us have written this second edition, the first edition had three additional authors—Bill Premerlani, Fred Eddy, and Bill Lorensen. We thank them for their past contribution on which this second edition builds. We also thank them for their support and encouragement in our writing of this second edition.

Chris Kelsey had an important role in the second edition that deserves special mention. She is the primary author of Chapter 18 on OO programming languages and also was an active reviewer.

We are grateful to our other reviewers (Mikael Berndtsson, Peter Chang, Bill Premerlani, and John Putnam) for taking the time to read our manuscript and provide thoughtful criticism.

Finally we wish to thank our families and colleagues for being patient with our many distractions and diversions during the writing of this book.

Michael Blaha
Chesterfield, Missouri
blaha@computer.org

James Rumbaugh
Cupertino, California

In addition there are a number of books that present the concepts of the UML. This book is different than most in that it not only explains the concepts, but it also explains their fundamental purpose and shows how to use them to build software. We do not explain every concept and nuance, but we do strive to explain the core of UML—enough to help you learn how to use the UML to build better software.

Changes From the First Edition

It has been fourteen years since we completed the first edition of this book. In the meantime there have been many advances in technology, leading to many changes in this second edition.

- **Notation**. We have replaced the OMT notation with the UML notation, specifically UML 2.0. The UML is now the dominant and standard language for OO modeling.
- **Process**. The second edition adds more content to the software development process. We now distinguish between domain analysis and application analysis. We have added implementation modeling. By intent, we have kept the process simple and lightweight so that it is approachable to students. This book's process is a subset of heavyweight processes, such as IBM Rational's RUP.
- **The three models**. We have carried forward the first edition's focus on "the three models" because we believe such an emphasis is helpful for teaching and learning OO modeling. However, we dropped the functional model, because it was not as useful as we had expected. In its place, we added the interaction model to incorporate use cases and sequence diagrams and to give a more holistic understanding of behavior among several objects.
- **Software engineering**. Part 4 covers several important software engineering topics: iterative development, management of models, and treatment of legacy systems.
- **Programming languages**. Programming languages have changed dramatically over the past decade and a half. Smalltalk has faded, while C and Fortran have diminished in importance. C++ and Java are now the dominant OO programming languages, and we have focused on them accordingly.
- **Databases**. OO models provide a sound basis not only for programming code, but also for relational databases. This book has an entire chapter that shows how to build efficient, correct, and extensible databases from UML models.
- **Case studies**. When the first edition was published, we felt a need to justify OO technology, so we included several case studies. Today, many case studies are available in the literature, so we have eliminated them from this book.

In this second edition, we have attempted to carry forward the first edition's style, emphasis on practical ideas, many examples, and many exercises.

cepts, such as syntax, semantics, recursion, set, procedure, graph, and state; a detailed formal background is not required.

Our emphasis differs from that of some in the object-oriented programming community but is in accord with the information modeling and design methodology communities. We emphasize object-oriented constructs as models of real things, rather than as techniques for programming. We elevate interobject relationships to the same semantic level as classes, rather than hiding them as pointers inside objects. We place somewhat less emphasis on inheritance and methods. We downplay fine details of inheritance mechanisms. We come down strongly in favor of typing, classes, modeling, and advance planning. We also show how to apply object-oriented concepts to state machines.

The book contains four parts. Part 1 presents object-oriented concepts in a high-level, language-independent manner. These concepts are fundamental to the rest of the book, although advanced material can be skipped initially. The UML notation is introduced in Part 1 and used throughout the book. Part 2 describes a step-by-step object-oriented methodology of software development from problem statement through analysis, system design, and class design. All but the final stages of the methodology are language independent. Part 3 describes the implementation of object-oriented designs in object-oriented languages and relational databases. It describes the considerations applicable to different environments, although it is not intended to replace books on object-oriented programming. Part 4 describes software engineering practices needed for successful object-oriented development.

The authors have used object-oriented analysis, design, programming, and database modeling for many years now on a variety of applications. We are enthusiastic about the object-oriented approach and have found it appropriate to almost any kind of application. We have found that the use of object-oriented concepts, together with a graphical notation and a development methodology, can greatly increase the quality, flexibility, and understandability of software. We hope that this book can help get that message across.

The book has a rich variety of exercises that cover a range of application domains and implementation targets. We suggest that you try working some of them as you go along. Ultimately, OO technology is not learned by reading about it, but by trying to practice it. Answers to selected exercises are included at the back of the book.

Comparison With Other Books

There are many books on the market that cover object-oriented technology. This book differs from most in that it teaches how to think about object-oriented modeling, rather than just presenting the mechanics of a programming language or modeling notation.

Many of the available object-oriented books are about programming issues, often from the point of view of a single language. Some of these books do discuss design issues, but they are still mainly about programming. Few books focus on object-oriented analysis or design. We show that object-oriented concepts can and should be applied throughout the entire software life cycle.

notation to be used throughout the entire software development process. The software developer does not need to translate into a new notation at each development stage.

We show how to use object-oriented concepts throughout the entire software life cycle, from analysis through design to implementation. The book is not primarily about object-oriented languages or coding. Instead we stress that coding is the last stage in a process of development that includes stating a problem, understanding its requirements, planning a solution, and implementing a program in a particular language. A good design technique defers implementation details until later stages of design to preserve flexibility. Mistakes in the front of the development process have a large impact on the ultimate product and on the time needed to finish. We describe the implementation of object-oriented designs in object-oriented languages and relational databases.

The book emphasizes that object-oriented technology is more than just a way of programming. Most importantly, it is a way of thinking abstractly about a problem using real-world concepts, rather than computer concepts. We have found this to be a difficult transition for some people. Books that emphasize object-oriented programming often fail to help the programmer learn to think abstractly. We have found that a graphical notation helps the software developer visualize a problem without prematurely resorting to implementation.

We show that object-oriented technology provides a practical, productive way to develop software for most applications, regardless of the final implementation language. We take an informal approach in this book; there are no proofs or formal definitions with Greek letters. We attempt to foster a pragmatic approach to problem solving by drawing upon the intuitive sense that object-oriented technology captures and by providing a notation and methodology for using it systematically on real problems. We provide tips and examples of good and bad design to help the software developer avoid common pitfalls.

Who Should Read This Book?

This book is intended for both software professionals and students. The reader will learn how to apply object-oriented concepts to all stages of the software development life cycle. We do not assume any prior knowledge of object-oriented concepts. We do assume that the reader is familiar with basic computing concepts, but an extensive formal background is not required. Even existing object-oriented programmers will benefit from learning how to design programs systematically; they may be surprised to discover that certain common object-oriented coding practices violate principles of good design.

The database designer will find much of interest here. Although object-oriented programming languages have received the most attention, object-oriented design of databases is also compelling and immediately practical. We include an entire chapter describing how to implement an object-oriented model using relational databases.

This book can be used as a textbook for a graduate or advanced undergraduate course on software engineering or object-oriented technology. It can be used as a supplementary text for courses on databases or programming languages. Prerequisites include exposure to modern programming languages and a knowledge of basic computer science terms and con-

Preface

Welcome to the second edition of *Object-Oriented Modeling and Design*. Much has changed since we finished the first book (1991). Back then object-oriented (OO) technology was considered new. Despite the excitement and enthusiasm, there was concern whether OO was really practical or just a passing fad. Consider all that has changed:

- **OO languages**. C++ is now established and Java has also become popular. The dominant programming languages are now OO.
- **OO databases**. Somewhat surprisingly, OO databases have faded, but relational databases are now including some OO features.
- **OO modeling**. The Unified Modeling Language (UML) standard from the Object Management Group has consolidated the multiple competing notations.
- **OO methodology**. Development methodologies now routinely incorporate OO ideas and concepts.

OO technology has truly become part of the computing mainstream. OO technology is no longer the exception; rather it is the usual practice.

What You Will Find

This book presents an object-oriented approach to software development based on modeling objects from the real world and then using the model to build a language-independent design organized around those objects. Object-oriented modeling and design promote better understanding of requirements, cleaner designs, and more maintainable systems. We describe a set of object-oriented concepts and a language-independent graphical notation that can be used to analyze problem requirements, design a solution to the problem, and then implement the solution in a programming language or database. Our approach allows the same concepts and

Contents

ISBN 978-81-317-1106-4

10 9 8

Published by Dorling Kindersley (India) Pvt. Ltd., licensees of Pearson Education in South Asia.

Head Office: 7th Floor, Knowledge Boulevard, A-8(A), Sector-62, Noida 201309, UP, India.
Registered Office: 11, Community Centre, Panchsheel Park, New Delhi 110 017, India.

Printed in India by Chennai Micro Print.

Object-Oriented Modeling and Design with UML™

Second Edition

Michael Blaha
Modelsoft Consulting Corporation

James Rumbaugh
IBM

PEARSON

What Others Have Said About Object-Oriented Modeling and Design with UML, Second Edition

"The first edition of *Object-Oriented Modeling and Design* by James Rumbaugh, Michael Blaha, and their colleagues is already a classic. It has influenced me more than any other book on modeling. I have successfully applied their ideas in large university project courses for over ten years now, and I am glad to see an updated version of this landmark book. It is bound to shape the thinking habits of another generation of software designers and modelers."

— *Bernd Bruegge, Technical University Munich*

"Blaha & Rumbaugh have done it again. They've updated their classic book for our current times, showing again that by their simple and straightforward explanations, their precise insights, and their key examples and exercises, that the adoption of object-oriented methodology need not be difficult. A must to have, read, and study by any practitioner."

— *Michael J. Chonoles*

"Our Master and Doctoral programs in information systems are adopting the *Object-Oriented Modeling and Design with UML* (OOMD) methodology. The book, written by two of the leading experts in the field, covers all aspects of OOMD with deep insight, many fine points, and up to date examples. It offers great value to our programs."

— *Peter H. Chang, Lawrence Technological University*

"If you are looking for a book that introduces UML, has a simple and useful object-oriented analysis and design process, and also includes details about important object-oriented concepts, then I strongly recommend that you study this excellent text."

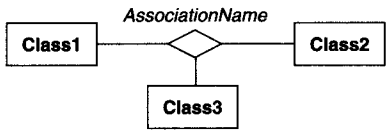
— *Mikael Berndtsson, University of Skövde*

Object-Oriented Modeling and Design with UML™

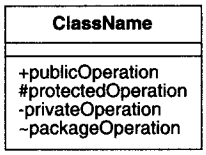
Second Edition

Class Model Notation — Advanced Concepts

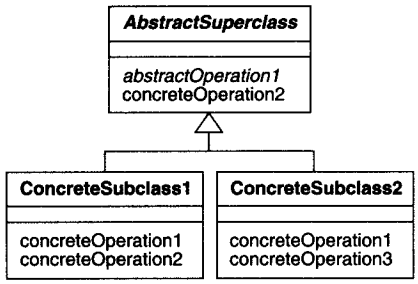
Ternary Association:



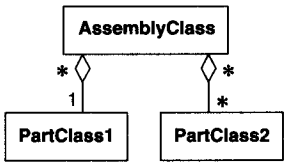
Visibility:



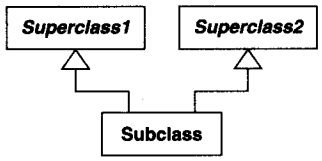
Abstract and Concrete Class:



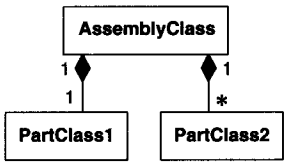
Aggregation:



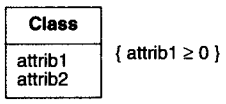
Multiple Inheritance, Disjoint:



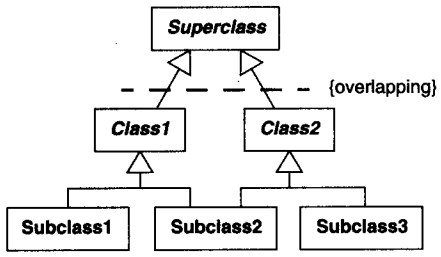
Composition:



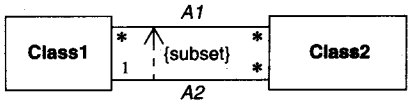
Constraint on Objects:



Multiple Inheritance, Overlapping:

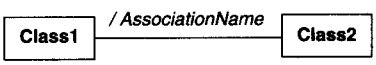


Constraint on Links:

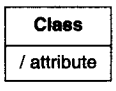


Derived Class:

Derived Association:

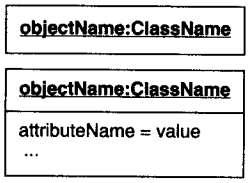


Derived Attribute:

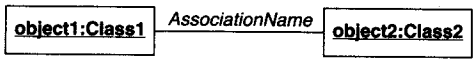


Class Model Notation — Basic Concepts

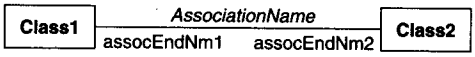
Object:



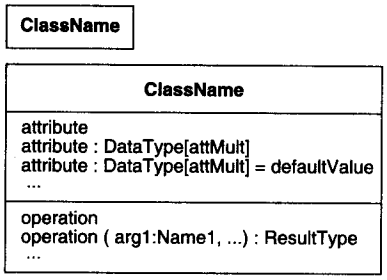
Link:



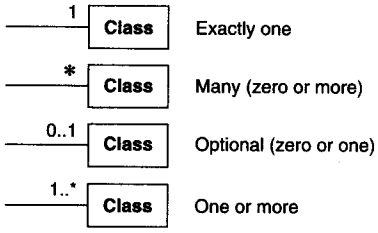
Association:



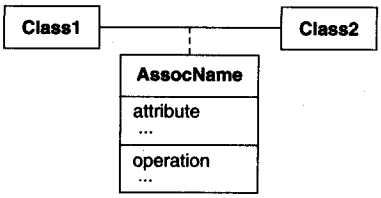
Class:



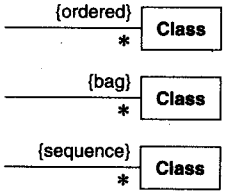
Multiplicity of Associations:



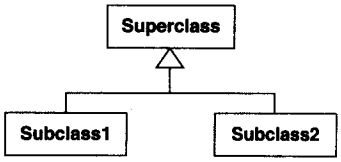
Association Class:



Ordered, Bag, Sequence:

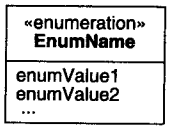


Generalization (Inheritance):

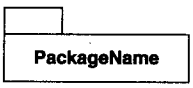


Qualified Association:

Enumeration:



Package:



Comment: