Object-Oriented Modeling and Design

James Rumbaugh
Michael Blaha
William Premerlani
Frederick Eddy
William Lorensen

Contents

PREFACE ix

Acknowledgments, xii

CHAPTER 1 INTRODUCTION 1

1.1 What Is Object-Oriented?, 1
1.2 What Is Object-Oriented Development?, 4
1.3 Object-Oriented Themes, 7
1.4 Evidence for Usefulness of Object-Oriented Development, 9
1.5 Organization of this Book, 10
Bibliographic Notes, 12
References, 12
Exercises, 13

Part 1: Modeling Concepts

CHAPTER 2 MODELING AS A DESIGN TECHNIQUE 15

2.1 Modeling, 15
2.2 The Object Modeling Technique, 16
2.3 Chapter Summary, 19
Exercises, 19

CHAPTER 3 OBJECT MODELING 21

3.1 Objects and Classes, 21
3.2 Links and Associations, 27
3.3 Advanced Link and Association Concepts, 31
3.4 Generalization and Inheritance, 38
3.5 Grouping Constructs, 43
3.6 A Sample Object Model, 43
3.7 Practical Tips, 46

3.8 Chapter Summary, 47
Bibliographic Notes, 48
References, 48
Exercises, 49

CHAPTER 4 ADVANCED OBJECT MODELING **57**

4.1 Aggregation, 57
4.2 Abstract Classes, 61
4.3 Generalization as Extension and Restriction, 63
4.4 Multiple Inheritance, 65
4.5 Metadata, 69
4.6 Candidate Keys, 71
4.7 Constraints, 73
4.8 Chapter Summary, 77
Bibliographic Notes, 79
References, 79
Exercises, 80

CHAPTER 5 DYNAMIC MODELING **84**

5.1 Events and States, 84
5.2 Operations, 92
5.3 Nested State Diagrams, 94
5.4 Concurrency, 99
5.5 Advanced Dynamic Modeling Concepts, 101
5.6 A Sample Dynamic Model, 105
5.7 Relation of Object and Dynamic Models, 110
5.8 Practical Tips, 111
5.9 Chapter Summary, 112
Bibliographic Notes, 113
References, 115
Exercises, 115

CHAPTER 6 FUNCTIONAL MODELING **123**

6.1 Functional Models, 123
6.2 Data Flow Diagrams, 124
6.3 Specifying Operations, 130
6.4 Constraints, 132
6.5 A Sample Functional Model, 133
6.6 Relation of Functional to Object and Dynamic Models, 137
6.7 Chapter Summary, 139
Bibliographic Notes, 140
References, 140
Exercises, 141

此为试读，需要完整PDF请访问：www.ertongbook.com

Part 2: Design Methodology

CHAPTER 7 METHODOLOGY PREVIEW **144**

7.1 OMT as a Software Engineering Methodology, 144
7.2 The OMT Methodology, 145
7.3 Impact of an Object-Oriented Approach, 146
7.4 Chapter Summary, 146
Exercises, 147

CHAPTER 8 ANALYSIS **148**

8.1 Overview of Analysis, 148
8.2 Problem Statement, 150
8.3 Automated Teller Machine Example, 151
8.4 Object Modeling, 152
8.5 Dynamic Modeling, 169
8.6 Functional Modeling, 179
8.7 Adding Operations, 183
8.8 Iterating the Analysis, 185
8.9 Chapter Summary, 187
Bibliographic Notes, 188
References, 188
Exercises, 189

CHAPTER 9 SYSTEM DESIGN **198**

9.1 Overview of System Design, 198
9.2 Breaking a System into Subsystems, 199
9.3 Identifying Concurrency, 202
9.4 Allocating Subsystems to Processors and Tasks, 203
9.5 Management of Data Stores, 205
9.6 Handling Global Resources, 207
9.7 Choosing Software Control Implementation, 207
9.8 Handling Boundary Conditions, 210
9.9 Setting Trade-off Priorities, 210
9.10 Common Architectural Frameworks, 211
9.11 Architecture of the ATM System, 217
9.12 Chapter Summary, 218
Bibliographic Notes, 220
References, 220
Exercises, 221

CHAPTER 10 OBJECT DESIGN **227**

10.1 Overview of Object Design, 227
10.2 Combining the Three Models, 229
10.3 Designing Algorithms, 230

10.4 Design Optimization, 235
10.5 Implementation of Control, 239
10.6 Adjustment of Inheritance, 242
10.7 Design of Associations, 245
10.8 Object Representation, 248
10.9 Physical Packaging, 249
10.10 Documenting Design Decisions, 251
10.11 Chapter Summary, 252
Bibliographic Notes, 254
References, 254
Exercises, 255

CHAPTER 11 METHODOLOGY SUMMARY **260**

11.1 Analysis, 261
11.2 System Design, 262
11.3 Object Design, 263
11.4 Chapter Summary, 264
Exercises, 264

CHAPTER 12 COMPARISON OF METHODOLOGIES **266**

12.1 Structured Analysis/Structured Design (SA/SD), 266
12.2 Jackson Structured Development (JSD), 268
12.3 Information Modeling Notations, 271
12.4 Object-Oriented Work, 273
12.5 Chapter Summary, 274
References, 275
Exercises, 275

Part 3: Implementation

CHAPTER 13 FROM DESIGN TO IMPLEMENTATION **278**

13.1 Implementation Using a Programming Language, 278
13.2 Implementation Using a Database System, 279
13.3 Implementation Outside a Computer, 280
13.4 Overview of Part 3, 280

CHAPTER 14 PROGRAMMING STYLE **281**

14.1 Object-Oriented Style, 281
14.2 Reusability, 282
14.3 Extensibility, 285
14.4 Robustness, 286
14.5 Programming-in-the-Large, 288
14.6 Chapter Summary, 291
Bibliographic Notes, 291

References, 292
Exercises, 292

CHAPTER 15 OBJECT-ORIENTED LANGUAGES 296

15.1 Translating a Design into an Implementation, 296
15.2 Class Definitions, 297
15.3 Creating Objects, 301
15.4 Calling Operations, 305
15.5 Using Inheritance, 308
15.6 Implementing Associations, 312
15.7 Object-Oriented Language Features, 318
15.8 Survey of Object-Oriented Languages, 325
15.9 Chapter Summary, 330
Bibliographic Notes, 332
References, 333
Exercises, 334

CHAPTER 16 NON-OBJECT-ORIENTED LANGUAGES 340

16.1 Mapping Object-Oriented Concepts, 340
16.2 Translating Classes into Data Structures, 342
16.3 Passing Arguments to Methods, 344
16.4 Allocating Objects, 345
16.5 Implementing Inheritance, 347
16.6 Implementing Method Resolution, 351
16.7 Implementing Associations, 355
16.8 Dealing with Concurrency, 358
16.9 Encapsulation, 359
16.10 What You Lose, 361
16.11 Chapter Summary, 362
Bibliographic Notes, 363
References, 364
Exercises, 364

CHAPTER 17 RELATIONAL DATABASES 366

17.1 General DBMS Concepts, 366
17.2 Relational DBMS Concepts, 368
17.3 Relational Database Design, 373
17.4 Advanced Relational DBMS, 387
17.5 Chapter Summary, 388
Bibliographic Notes, 389
References, 390
Exercises, 390

Part 4: Applications

CHAPTER 18 OBJECT DIAGRAM COMPILER 397

18.1 Background, 398
18.2 Problem Statement, 399
18.3 Analysis, 401
18.4 System Design, 407
18.5 Object Design, 408
18.6 Implementation, 412
18.7 Lessons Learned, 412
18.8 Chapter Summary, 413
Bibliographic Notes, 413
References, 413
Exercises, 414

CHAPTER 19 COMPUTER ANIMATION 416

19.1 Background, 417
19.2 Problem Statement, 418
19.3 Analysis, 420
19.4 System Design, 424
19.5 Object Design, 426
19.6 Implementation, 428
19.7 Lessons Learned, 430
19.8 Chapter Summary, 431
Bibliographic Notes, 431
References, 432
Exercises, 432

CHAPTER 20 ELECTRICAL DISTRIBUTION DESIGN SYSTEM 433

20.1 Background, 433
20.2 Problem Statement, 435
20.3 Analysis, 436
20.4 System Design, 444
20.5 Object Design, 445
20.6 Implementation, 448
20.7 Lessons Learned, 448
20.8 Chapter Summary, 449
Bibliographic Notes, 449
References, 449
Exercises, 450

APPENDIX A OMT GRAPHICAL NOTATION 453
APPENDIX B GLOSSARY 454
ANSWERS TO SELECTED EXERCISES 465
INDEX 491

此为试读,需要完整PDF请访问: www.ertongbook.com

Preface

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, the Object Modeling Technique, 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 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 as is required by many other methodologies.

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, non-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. This may be a difficult transition for some people because older programming languages force one to think in terms of the computer

and not in terms of the application. Books that emphasize object-oriented programming often fail to help the programmer learn to think abstractly without using programming constructs. We have found that the graphical notation that we describe 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 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. To illustrate the pragmatic nature of these concepts, we describe several real applications developed by the authors using object-oriented techniques.

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. At present, there are few, if any, object-oriented books covering the entire life cycle, as opposed to programming or analysis alone. In fact, there are few textbooks on object-oriented technology of any kind. Although object-oriented technology is currently a "hot" topic, most readers have limited experience with it, so 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 previously received the most attention, object-oriented design of databases is perhaps even more compelling and immediately practical. We include an entire chapter describing how to implement an object-oriented design using existing relational database management systems.

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 structured programming languages and a knowledge of basic computer science terms and concepts, such as syntax, semantics, recursion, set, procedure, graph, and state; a detailed formal background is not required. Exercises of varying difficulty are included in each chapter along with selected answers at the back of the book.

Many object-oriented books primarily discuss programming issues, usually from the point of view of a single language. The best of them discuss design issues, but they are nevertheless mainly about programming. Fewer books address object-oriented analysis or design. We show that object-oriented concepts can and should be applied throughout the entire software life cycle. Recently books on object-oriented methodology have begun to appear. Our book is compatible with other books on object-oriented analysis and design, and we feel that it is complementary to them in content.

Several existing books on software methodology discuss the entire life cycle from a procedural viewpoint. The traditional data flow methodologies of DeMarco, Yourdon, and others are based mainly on functional decomposition, although recent revisions have been influenced by object-oriented concepts. Even Jackson's methodology, which superficially seems to be based on objects, quickly reverts to procedural issues.

Our emphasis differs in some respects from the majority of the object-oriented programming community but is in accord with the information modeling and design methodology communities. We place a much greater emphasis on 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 importance 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 use terminology that is universally accepted when possible, otherwise we try to choose the best terms among various alternatives. There is as yet no commonly accepted graphical notation for object-oriented constructs, so despite concerns about introducing "yet another notation" we use our own Object Modeling Technique notation, which we have used extensively on real problems and which has been successfully adopted by others. In any case, the object-oriented concepts themselves are the most important thing, not the shape of the symbols used to represent them. 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 Object Modeling Technique notation is introduced in Part 1 and used throughout the book to show examples. Part 2 describes a step-by-step object-oriented methodology of software development from problem statement through analysis, system design, and object design. All but the final stages of the methodology are language-independent; even object design is concerned mostly with issues independent of any particular language. Part 3 describes the implementation of object-oriented designs in various target environments, including object-oriented languages, non-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 presents case studies of actual object-oriented applications developed by the authors at the General Electric Research and Development Center. The problems cover a range of application domains and implementation targets.

The authors have used object-oriented analysis, design, programming, and database modeling for several years on a variety of applications. We have also implemented an object-oriented language, developed an object-oriented notation and methodology, and developed object-oriented support tools, so we are familiar with both theoretical and pragmatic issues of implementing and using object-oriented technology. We are enthusiastic about the object-oriented approach and have found that it is applicable 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 to get that message across.

ACKNOWLEDGMENTS

We wish to thank the many individuals who have made this book possible. We especially want to thank GE and our management at the Research and Development Center for their foresight in giving us the opportunity to develop the ideas presented here by working on object-oriented technology when it was still a new and unproven field as well as for their support, encouragement, and facilities in writing the book. We also wish to thank our colleagues at GE who worked with us in exploring this exciting new field. We acknowledge the important contribution of Mary Loomis and Ashwin Shah, who participated in the original development of the Object Modeling Technique notation.

Many individuals helped in the review of the manuscript, but in particular we wish to thank David Hentchel, Mark Kornfein, and Marc Laymon for their thorough reviews and perceptive comments.

Finally and most importantly we wish to thank our wives and families for their patience and encouragement during the many long weekends and evenings that went into the writing of this book.

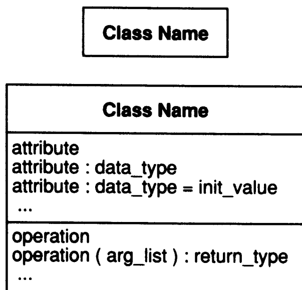
Production Note

The manuscript of this book was prepared by the authors on SUN workstations using the FrameMaker document preparation system. We drew the diagrams using the FrameMaker system. We created most object diagrams using our OMTool editor and converted them to FrameMaker format. We performed detailed page layout, made the index, and generated the table of contents using the FrameMaker system. Proof copies of the complete document were printed on Apple LaserWriter Plus printers. We generated PostScript page description files from the final document, copied them onto a Unix *tar* tape, and sent the tape to the publisher for generation of the camera copy on a Linotronic 202 typesetter. The publisher prepared and set the title and copyright pages.

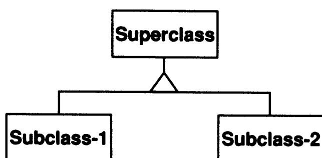
此为试读，需要完整PDF请访问：www.ertongbook.com

Object Model Notation
Basic Concepts

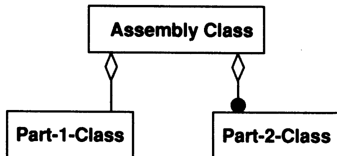
Class:



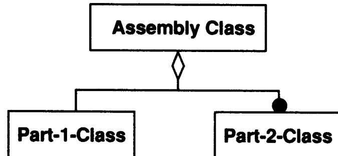
Generalization (Inheritance):



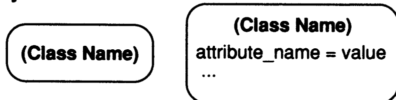
Aggregation:



Aggregation (alternate form):

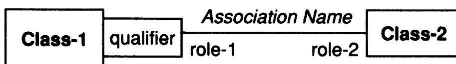


Object Instances:

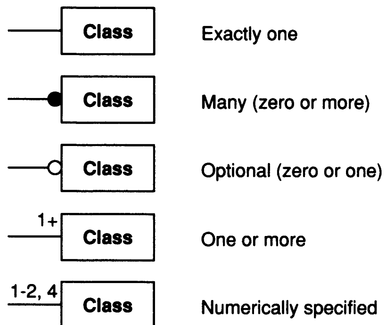


Association:

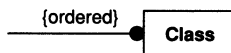
Qualified Association:



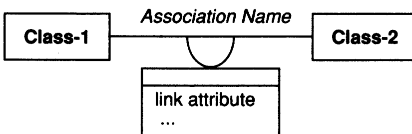
Multiplicity of Associations:



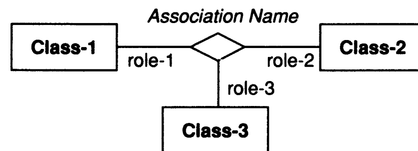
Ordering:



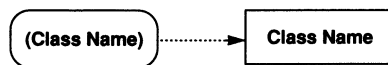
Link Attribute:



Ternary Association:



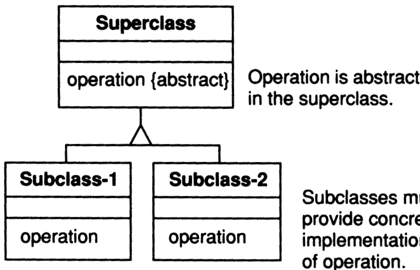
Instantiation Relationship:



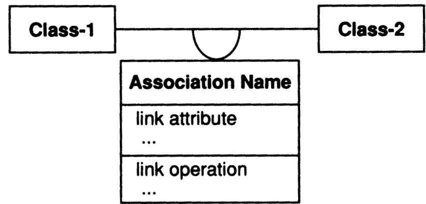
This page may be freely copied without obtaining permission from the publisher.

Object Model Notation
Advanced Concepts

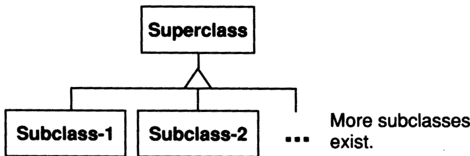
Abstract Operation:

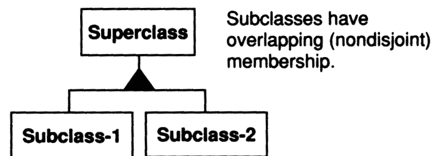


Association as Class:

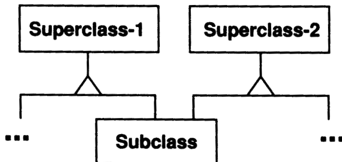


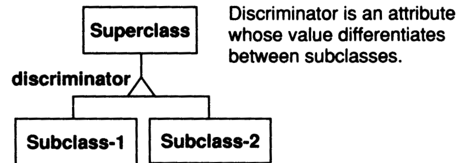
Generalization Properties:



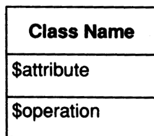


Multiple Inheritance:

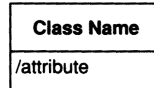




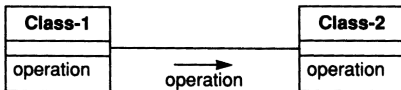
Class Attributes and Class Operations:



Derived Attribute:



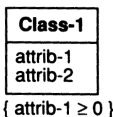
Propagation of Operations:



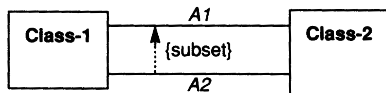
Derived Class:

Derived Association:

Constraints on Objects:



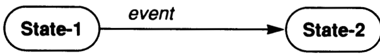
Constraint between Associations:



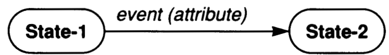
This page may be freely copied without obtaining permission from the publisher.

Dynamic Model Notation

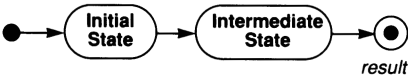
Event causes Transition between States:



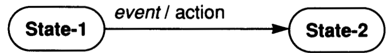
Event with Attribute:



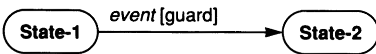
Initial and Final States:



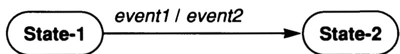
Action on a Transition:



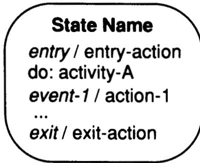
Guarded Transition:



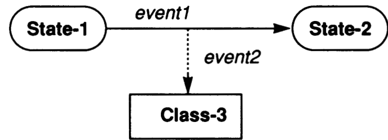
Output Event on a Transition:



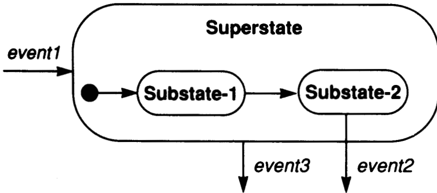
Actions and Activity while in a State:



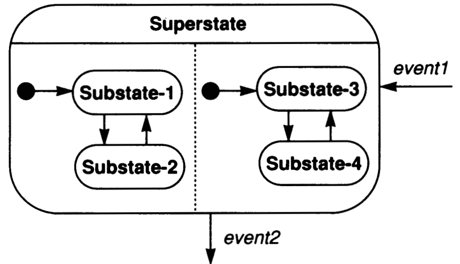
Sending an event to another object:



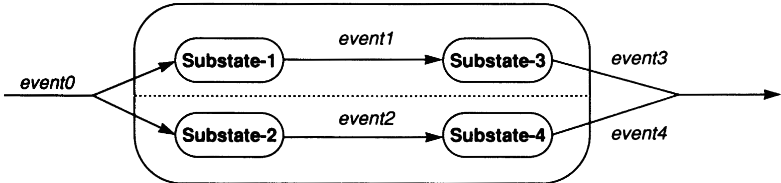
State Generalization (Nesting):



Concurrent Subdiagrams:



Splitting of control: **Synchronization of control:**

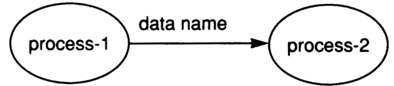


This page may be freely copied without obtaining permission from the publisher.

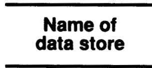
Functional Model Notation

Process:

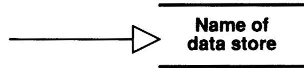
Data Flow between Processes:



Data Store or File Object:

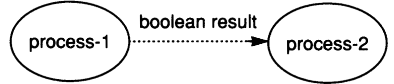


Data Flow that Results in a Data Store:

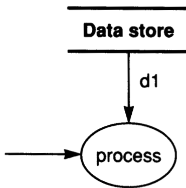


Actor Objects (as Source or Sink of Data):

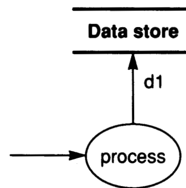
Control Flow:



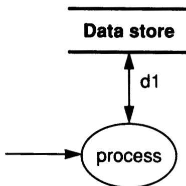
Access of Data Store Value:



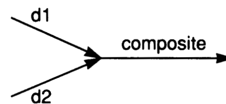
Update of Data Store Value:



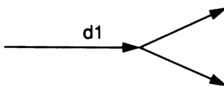
Access and Update of Data Store Value:



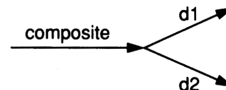
Composition of Data Value:



Duplication of Data Value:



Decomposition of Data Value:



This page may be freely copied without obtaining permission from the publisher.

1

Introduction

Object-oriented modeling and design is a new way of thinking about problems using models organized around real-world concepts. The fundamental construct is the object, which combines both data structure and behavior in a single entity. Object-oriented models are useful for understanding problems, communicating with application experts, modeling enterprises, preparing documentation, and designing programs and databases. This book presents an object-oriented software development methodology, the Object Modeling Technique (OMT), which extends from analysis through design to implementation. First an analysis model is built to abstract essential aspects of the application domain without regard for eventual implementation. This model contains objects found in the application domain, including a description of the properties of the objects and their behavior. Then design decisions are made and details are added to the model to describe and optimize the implementation. The application-domain objects form the framework of the design model, but they are implemented in terms of computer-domain objects. Finally the design model is implemented in a programming language, database, or hardware.

We describe a graphical notation for expressing object-oriented models. Application-domain and computer-domain objects can be modeled, designed, and implemented using the same object-oriented concepts and notation. The same seamless notation is used from analysis to design to implementation so that information added in one stage of development need not be lost or translated for the next stage.

1.1 WHAT IS OBJECT-ORIENTED?

Superficially the term "object-oriented" means that we organize software as a collection of discrete objects that incorporate both data structure and behavior. This is in contrast to conventional programming in which data structure and behavior are only loosely connected. There is some dispute about exactly what characteristics are required by an object-oriented approach, but they generally include four aspects: identity, classification, polymorphism, and inheritance.

1.1.1 Characteristics of Objects

Identity means that data is quantized into discrete, distinguishable entities called *objects*. A *paragraph in a document*, a *window on my workstation*, and the *white queen in a chess game* are examples of objects. Figure 1.1 shows some additional objects. Objects can be concrete, such as a *file* in a file system, or conceptual, such as a *scheduling policy* in a multiprocessing operating system. Each object has its own inherent identity. In other words, two objects are distinct even if all their attribute values (such as name and size) are identical.

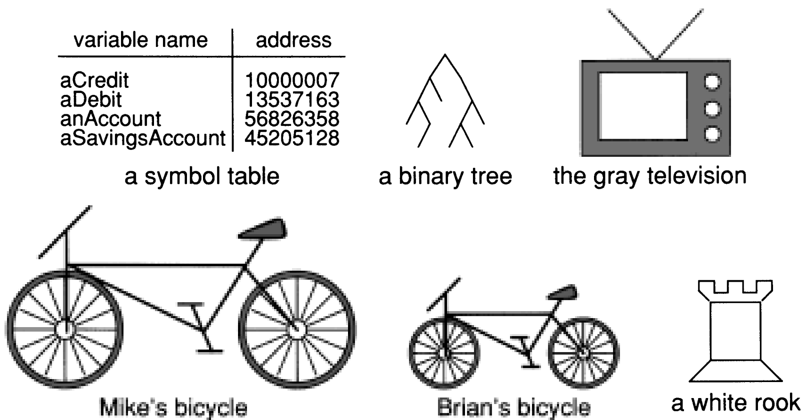


Figure 1.1 Objects

In the real world an object simply exists, but within a programming language each object has a unique *handle* by which it can be uniquely referenced. The handle may be implemented in various ways, such as an address, array index, or unique value of an attribute. Object references are uniform and independent of the contents of the objects, permitting mixed collections of objects to be created, such as a file system directory that contains both files and subdirectories.

Classification means that objects with the same data structure (*attributes*) and behavior (*operations*) are grouped into a *class*. *Paragraph*, *Window*, and *ChessPiece* are examples of classes. A *class* is an abstraction that describes properties important to an application and ignores the rest. Any choice of classes is arbitrary and depends on the application.

Each class describes a possibly infinite set of individual objects. Each object is said to be an *instance* of its class. Each instance of the class has its own value for each attribute but shares the attribute names and operations with other instances of the class. Figure 1.2 shows two classes and some of their respective instance objects. An object contains an implicit reference to its own class; it "knows what kind of thing it is."

Polymorphism means that the same operation may behave differently on different classes. The *move* operation, for example, may behave differently on the *Window* and *ChessPiece* classes. An *operation* is an action or transformation that an object performs or is subject to. *Right-justify*, *display*, and *move* are examples of operations. A specific implementation of an

此为试读，需要完整PDF请访问：www.ertongbook.com