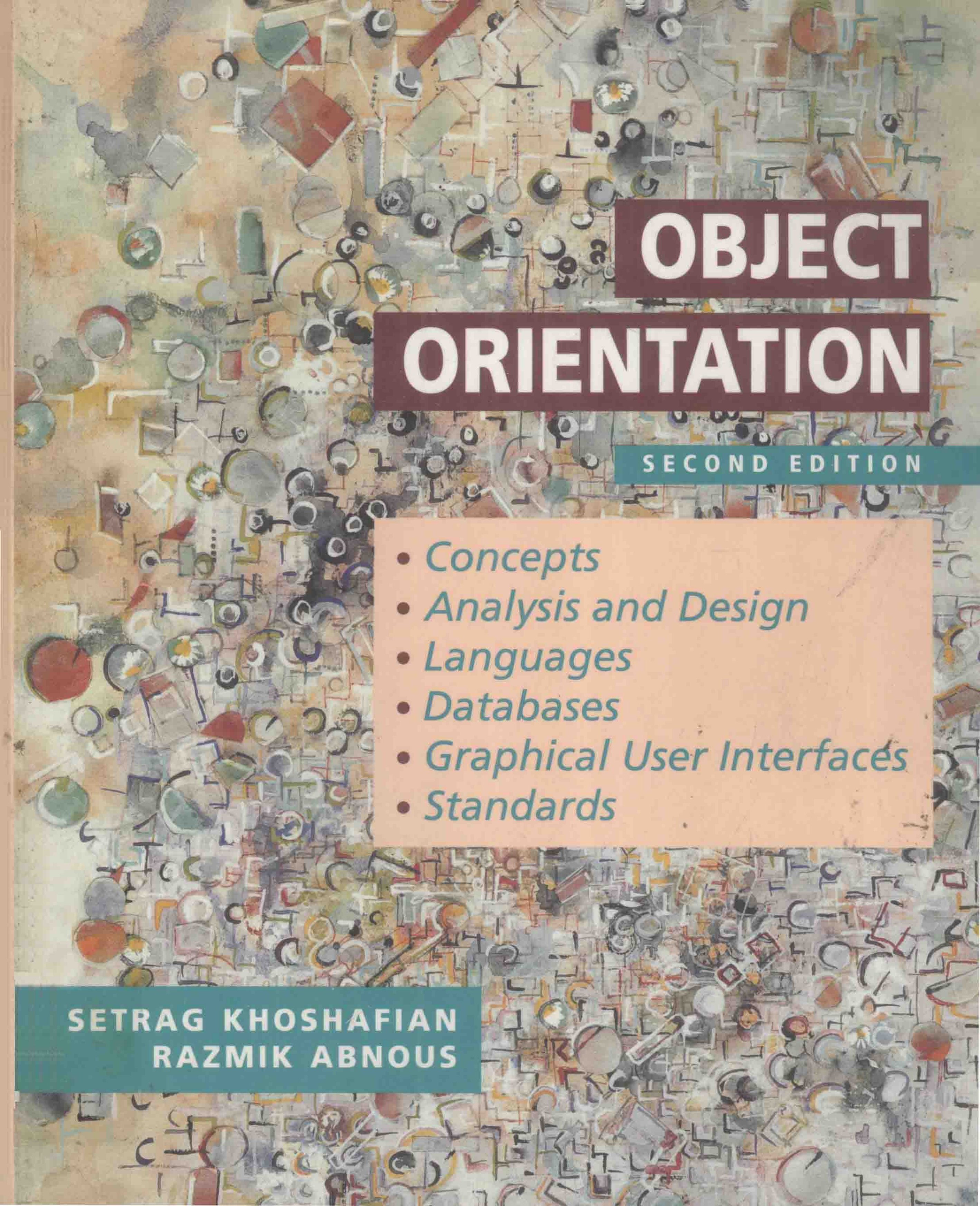
OBJECT ORIENTATION

SECOND EDITION

- *Concepts*
- *Analysis and Design*
- *Languages*
- *Databases*
- *Graphical User Interfaces*
- *Standards*

SETRAG KHOSHAFIAN
RAZMIK ABNOUS



Object Orientation
Second Edition

CONCEPTS • ANALYSIS & DESIGN • LANGUAGES • DATABASES • GRAPHICAL USER INTERFACES • STANDARDS

Setrag Khoshafian
Razmik Abnous

John Wiley & Sons, Inc.
New York • Chichester • Brisbane • Toronto • Singapore

Publisher: Katherine Schowalter
Senior Editor: Diane D. Cerra
Managing Editor: Micheline Frederick
Text Design & Composition: North Market Street Graphics

Library of Congress Cataloging-in-Publication Data
Khoshafian, Setrag.
Object orientation : concepts, analysis & design, languages, databases, graphical user interfaces, standards / Setrag Khoshafian, Razmik Abnous.
p. cm.
Includes bibliographical references.
ISBN 0-471-07834-4 (pbk.)
1. Object-oriented programming (Computer science) I. Abnous, Razmik.
II. Title.
QA76.64.K47 1995
005.1'1—dc20 95-22368
CIP

Printed in the United States of America
10 9 8 7 6 5 4 3 2 1

Object Orientation
Second Edition

To our wives
Silva Khoshafian and
Suzanne M. Abnous

and children
Nishan, Jonathan, Shahan, and Nareg Khoshafian,
Sevan, Haig, and Armen Abnous

Preface

Application environments are becoming increasingly fast-paced and complex. There is great diversity in the way information is accessed, manipulated, and presented. Information is no longer centralized; it is distributed on networks of heterogeneous environments. Different generic products and tools attempt to simplify a user's interactions with information systems. Unfortunately, the diversity and the lack of integration among these products often force users to learn multiple-user interface paradigms, obtuse operating system scripts, and different key-stroke commands.

There is an enormous need for integration, simplicity, and ease of use across all products. But simplicity and ease of use come at a price. The simpler and more intuitive the environment, the more complex the underlying software system must be to support and implement it. To meet the computational needs of emerging applications, software must be modular, extensible, maintainable, and robust. It must be manageable. This means the model of the real world and its implementation in the system should not be too far off.

Object orientation is an enabling software technology that attempts to fulfill these increasingly demanding computational needs. The potential of object technology stems from:

1. The proficiency of its higher-level object-oriented model, which also reduces software cost and provides the designer with real-world programmable components.
2. Its capability to share and reuse code with object-oriented engineering techniques, reducing the time required to develop an application.

3. Its capability to localize and minimize the effects of modifications through object-oriented programming abstractions, allowing the software to encompass more diverse algorithms and technologies.

This book clarifies the basic concepts associated with object orientation. It provides a clear yet comprehensive exposure to the fundamental ideas in object orientation. This is important since object orientation is often confused with certain languages, such as Smalltalk, or particular user-interface paradigms, such as windows and icons. Furthermore, a clear understanding of object-oriented concepts will allow the reader to use object-oriented constructs in *any* programming language (including conventional ones).

The book will give the reader a basic understanding of the main concepts of popular object-oriented systems: languages, databases, and user interfaces. It also elucidates object-oriented analysis and design methodologies and shows how the object-oriented concepts manifest themselves in object-oriented software development. Object orientation is above all a programming *style* that allows better organization and modularization of large application programs. The book emphasizes the concepts and the ideas for improving programming style, regardless of whether the programmer uses an object-oriented programming language. Special emphasis is given to C++ and to the newly emerging fields of object-oriented analysis and design, object-oriented databases, and user interfaces.

Many programming languages are labeled or claim to be "object-oriented." We felt it was important to show how object-oriented concepts are reflected in these languages. Simula and Smalltalk have played a historic role in the evolution of object-oriented systems and concepts. Nevertheless, we believe the most popular system development languages of the 1990s will be C++. Hence we devote an entire chapter to it.

The book also demonstrates the object-oriented concepts that manifest themselves in database applications. We felt the need to present a clear exposure of the emerging field of object-oriented databases. These databases constitute the next evolutionary step after relational databases; post–relational database management systems will all have object-oriented features. This book demonstrates both the object-oriented characteristics and database characteristics of several systems that claim to be object-oriented databases.

As the emergence of Windows 95 and the popularity of graphical user interfaces (GUI) in other platforms (MacOS, Windows X, and OS/2 Warp) indicate, user interaction with computing systems is becoming more intuitive. Given the explosion in multimedia information management systems and graphical information displays, we decided to analyze the object-oriented capabilities of graphical user interfaces. These GUI environments provide direct object manipulation features. They allow users to represent and interact with visual objects using familiar physical metaphors. This book uncovers the object-oriented characteristics of the most popular GUI systems.

This book is intended for a wide variety of audiences. At small and large corporations, MIS managers and professionals can use the book to understand the impact of object orientation on their end users. The book will serve the needs of software engineers and computer scientists who are interested in object-oriented systems and want a global perspective on the subject. At universities, it could be a supplementary text for programming languages and software engineering courses, or a graduate course dedicated to object-oriented systems. The book is also suitable for short courses, continuing education, and professional self-study. Prerequisites in data structures, compilers, operating systems, performance evaluation, and databases will be useful.

Chapter 1 gives a brief introduction to object-oriented concepts. It examines the evolution of object orientation in languages, databases, and user interfaces. It also presents examples used commonly throughout the book.

The key object-oriented concepts covered in this book are abstract data typing, inheritance, and object identity.

Chapters 2 through 4 provide detailed and self-contained exposure to all the aspects of these foundational concepts.

Chapter 2 discusses the concept of abstract data typing. It describes in detail the mechanisms of hiding the implementation of an object's interface routines. Abstract data typing enhances code extensibility and reusability. The chapter also clarifies and differentiates commonly used object-oriented terms, such as *classes, instances, methods,* and *messages.*

Chapter 3 discusses the concept of inheritance. Through inheritance, one can build new classes or software modules on top of an existing, less specialized hierarchy of classes, instead of redesigning everything from scratch. The new classes can inherit both the behavior and the representation of existing classes. Inheritance enhances software extensibility, reusability, and code sharing.

Chapter 4 surveys the concept of object identity. Object identity allows the objects in an application to be organized in arbitrary graph-structured object spaces. The chapter shows the superiority and generality of the strong notion of object identity when compared to the conventional techniques of referencing objects in programming languages and databases. Three disciplines have been most influential in the evolution of object-oriented technologies; object-oriented languages, object-oriented databases, and object-oriented user interfaces.

Chapter 5 provides an overview of object-oriented analysis and design methodologies. The chapter demonstrates how a step-by-step software design and implementation methodology can be used to construct powerful applications with object technology. It also provides brief descriptions of a number of popular OOA×OOD methodologies, including Booch, Rumbaugh, and Shlaer/Mellor.

Throughout Chapters 2, 3, and 4 are numerous illustrative examples in some of the most common object-oriented languages, such as Smalltalk. However, we dedicate two chapters to popular object-oriented languages. Chapter 6 discusses languages such as Smalltalk, Ada, and Eiffel. We have chosen Smalltalk since it started the object-oriented revolution, Ada because of its popularity in government, and Eiffel since it represents a modern object-oriented language. Chapter 7 concentrates on C++, describing its main features through illustrative examples. C++ already is *the* system development language of the 1990s. The chapter, which is self-contained, will provide the reader with a firm grasp of the language.

Chapter 8 covers the emerging field of object-oriented databases, which combine the object-oriented concepts of abstract data typing, inheritance, and object identity with database capabilities. These database capabilities include the support of persistent object spaces, transactions, recovery, querying, and versioning of objects. The chapter discusses the object-oriented features of several object-oriented database management systems from both research and industry.

Chapter 9 illustrates the enormous impact of object orientation on the design and presentation of the modern user interfaces. This chapter covers such concepts as the direct manipulation of objects and the application of predefined class hierarchy libraries for user-interface components. It also discusses some of the modern graphical user interfaces in use today and describes the object-oriented interface layers presented to both end users and application developers.

Chapter 10 presents an overview of standardization efforts in object technologies. In this chapter we examine standardization in the areas of object sharing and interoperability and databases. Object sharing and interoperability have become very hot topics in the 1990s. We examine Microsoft OLE2, CILab's OpenDoc, and Object Management Group's CORBA. In the area of databases, we examine the ODMG-93 standard, which provides a framework for interoperability and sharing among object-oriented database vendors.

ACKNOWLEDGMENTS

We would like to extend our appreciation to people who helped us with this book. Above all we want to thank our wives—Silva Khoshafian and Suzanne M. Abnous—for their support and patience. We would also like to express our gratitude to our editors: Diane Cerra and Tammy Boyd. We are grateful to all those who helped us with the first edition of our book—and contributed to its success.

Walnut Creek, California
Danville, California
July, 1995

SETRAG KHOSHAFIAN
RAZMIK ABNOUS

Contents

1

INTRODUCTION 1

1.1 Introduction to Object Orientation 1
  1.1.1 Object Orientation for All Users 6
  1.1.2 What Is Object Orientation? 7
    1.1.2.1 Abstract Data Types 9
    1.1.2.2 Inheritance 9
    1.1.2.3 Object Identity 9
    1.1.2.4 Object-Oriented Analysis and Design 10
  1.1.3 Programming Languages, Databases, and Interfaces 12

1.2 History and Evolution 13
  1.2.1 The Evolution of Object Orientation in Programming Languages 13
  1.2.2 Evolution of Object-Oriented Databases 19
    1.2.2.1 Network and Hierarchical Models 19
    1.2.2.2 The Relational Model 20
    1.2.2.3 Semantic and Functional Data Models 21
    1.2.2.4 Extended and Object-Relational Models 22
    1.2.2.5 Nested Relational and Complex Object Models 22
  1.2.3 The Evolution of Object Orientation in User Interfaces 25

1.3 Summary 30

2

ABSTRACT DATA TYPES 33

2.1 Introduction 33
2.1.1 Data Types 34
2.1.2 From Data Types to Abstract Data Types 35
2.1.3 The Object/Message Paradigm 39
2.1.4 Modularization through Procedures versus Objects 41
2.1.4.1 Modeling of the Real World 41
2.1.4.2 Autonomy 41
2.1.4.3 Generation of Correct Applications 43
2.1.4.4 Reusability 43
2.1.5 Summarizing the Benefits of Abstract Data Typing 44
2.1.6 Chapter Organization 44

2.2 Classes 44
2.2.1 Instance Values 47
2.2.2 Methods and Messages 50
2.2.2.1 The Method Body 51
2.2.2.2 Implicit Parameters and the Pseudovariable "Self" 52
2.2.2.3 Accessor and Update Methods 53
2.2.3 Creating and Destroying Objects 54
2.2.3.1 Creating Objects in Object-Oriented Languages 56
2.2.3.2 Destroying Objects and Garbage Collection in Object-Oriented Languages 57
2.2.3.3 Garbage Collection Strategies and Algorithms 59
2.2.4 Class Extensions 66
Collections 67

2.3 Overloading 69

2.4 Dynamic Binding 69

2.5 Parametric Polymorphism or Genericity 72
2.5.1 Parametric Abstract Data Types 73
2.5.2 Just Syntactic Sugar? 74

2.6 Constraints 75
2.6.1 Constraints on Object and Instance Variables 76
2.6.2 Pre-and Postconditions of Methods 76

2.7 Summary 77

3

INHERITANCE 78

3.1 Introduction 78
3.1.1 Inheritance in Knowledge Representation 78
3.1.2 Inheritance in Object-Oriented Languages 80
3.1.3 The Different Facets of Inheritance 82
3.1.4 Chapter Organization 85
3.2 Inheritance and Subtyping 85
3.2.1 Subtyping 85
3.2.1.1 Subsets as Subtypes 86
3.2.1.2 Subtyping of Structured Types 88
3.2.2 Contrasting Inheritance with Subtyping 89
3.2.2.1 Implicit Subtyping versus Explicit Inheritance 91
3.2.2.2 Subtyping and Dynamic Binding 92
3.2.2.3 What Do Classes Inherit? 94
3.3 Class Inheritance 95
3.3.1 Inheriting Instance Variables 98
3.3.1.1 Redefining Instance Variables 98
3.3.1.2 Hiding Instance Variables 100
3.3.2 Inheriting Methods 104
3.3.2.1 Method Overriding 105
3.3.2.2 Invoking Superclass Methods 107
3.3.2.3 Constrained Overriding 108
3.3.2.4 Inheriting the Interface 113
3.3.2.5 Excluding Superclass Methods 114
3.4 Metaclasses 116
3.4.1 Explicit Support 117
3.4.2 Implicit or Hidden Metaclasses 120
3.4.3 Object-Oriented Languages without Metaclasses 122
3.5 Object Inheritance 123
3.5.1 Prototype Systems and Delegation 125
3.6 Multiple Inheritance 130
3.6.1 Linearization 134
3.6.2 Forbidding Conflicts—Renaming Strategies 135
3.6.3 Qualified Instance Variable and Methods 136
3.6.4 The Meet Operation 137
3.6.5 Evaluating the Strategy 138
3.7 Summary 138

4

OBJECT IDENTITY 140

4.1 Introduction 140
4.1.1 So What's the Big Deal about Object Identity? 140
4.1.2 Chapter Organization 141
4.2 Referencing Objects in Programming Languages 141
4.3 Object Naming 147
4.3.1 Path Names in Operating Systems 147
4.4 Identity Through Identifier Keys 149
4.5 The Type/State/Identity/ Trichotomy 151
4.5.1 Object Spaces with Identity 154
4.5.2 Implementation Techniques 158
4.5.2.1 Persistence 158
4.5.2.2 The Implementation Strategies 160
4.6 Operations with Identity 164
4.6.1 The Different Facets of Equality 165
4.6.2 The Different Facets of Copying 170
4.6.3 Merging and Swapping 172
4.7 Summary 175

5

OBJECT-ORIENTED ANALYSIS AND DESIGN 176

5.1 Introduction 176
5.1.1 Modeling Software Development Process 178
5.1.2 Object-Oriented Analysis and Design 180
5.1.3 Object-Oriented Programming: How Much Should We Design? 180
5.1.4 Too Many Methodologies! 182
5.2 Invention, Aspirations, and Requirements 183
5.3 Object-Oriented Analysis 184
5.4 Defining Object-Oriented Design 188
5.4.1 Coarse-Grained Design 189
5.4.2 More Detailed Design 190
5.4.3 Design Specifications 192

5.5 Semantic and Entity Relationship Modeling 194
5.5.1 Semantic Data Modeling Background 194
5.5.2 Entity Relationship Modeling 197
5.5.3 Contrasting Design for Databases and OOA/OOD 200
5.5.4 Double Design 201
5.6 Overview of Existing Methodologies 202
5.6.1 Booch's OOA and OOD Approach 204
5.6.1.1 Requirement Analysis 204
5.6.1.2 Domain Analysis 204
5.6.1.3 Object-Oriented Design 206
5.6.2 The OMT OOA and OOD Approach 210
5.6.2.1 Object-Oriented Analysis 210
5.6.2.2 Object-Oriented Design 211
5.6.2.3 Object Diagrams 211
5.6.2.4 Dynamic Model 213
5.6.2.5 Functional Model 214
5.6.3 Shlaer and Mellor OOA and OOD Method 215
5.6.3.1 Partitioning into Domains 216
5.6.3.2 Objects and Information Model 217
5.6.3.3 Lifecycles and State Transition Diagrams 218
5.6.3.4 Object Communication Development 219
5.6.3.5 Action Data Flow Diagrams 219
5.6.3.6 OOD in Shlae•and Mellor 221
5.7 Summary 222

6

OVERVIEW OF OBJECT-ORIENTED LANGUAGES 223

6.1 Introduction 223
6.2 Small Talk 223
6.2.1 Abstract Data Typing 224
6.2.1.1 Object Creation and Manipulation 224
6.2.1.2 Class Definition 225
6.2.1.3 The Object Class 227
6.2.2 Inheritance 228
6.2.2.1 *self* and *super* Pseudovariables 229
6.2.2.2 Abstract Classes 230
6.2.3 Polymorphism 230
6.2.3.1 Method Overriding 231
6.2.3.2 Operator Overloading 231
6.2.3.3 Parametric Polymorphism 232

6.2.4 Object Identity 233
6.2.5 Container Classes 234

6.3 Ada 235
6.3.1 Object-Oriented Concepts in Ada 235
6.3.2 Types 235
6.3.2.1 Subtypes Definition 237
6.3.2.2 Derived Type Definition 238
6.3.2.3 Type Discriminants 239
6.3.3 Abstract Data Typing and Packages 240
6.3.3.1 Package Specification 240
6.3.3.2 Referencing Packages 241
6.3.3.3 Package Body 243
6.3.3.4 Private Type Declaration 243
6.3.4 Polymorphism 244
6.3.4.1 Overloading 244
6.3.4.2 Parametric Polymorphism and Generic Units 246
6.3.5 Ada 9X 251
6.3.5.1 Inheritance 251
6.3.5.2 Dynamic Binding 254
6.3.5.3 Abstract Classes 254

6.4 Eiffel 255
6.4.1 Abstract Data Typing 255
6.4.1.1 Object Creation 256
6.4.1.2 Information Hiding 256
6.4.1.3 Assertions—Pre/Postconditions 257
6.4.2 Inheritance 259
6.4.2.1 Export Status 259
6.4.2.2 Feature Adaptation 259
6.4.2.3 Repeated Inheritance 262
6.4.2.4 Deferred Class 263
6.4.3 Polymorphism 263
6.4.3.1 Overloading 264
6.4.3.2 Generic Classes 264
6.4.4 Object Identity 266

6.5 Summary 266

7

C++ **267**

7.1 Introduction 267
7.1.1 Chapter Organization 267

7.2 Object Orientation in C++ 268

7.3 Pros and Cons of C++ 269
- 7.3.1 Advantages 269
- 7.3.2 Disadvantages 269

7.4 Class Definition 270
- 7.4.1 Class Construct 270
  - 7.4.1.1 Member Function 271
  - 7.4.1.2 The *this* Pointer 272
- 7.4.2 Constructors and Destructors 272
  - 7.4.2.1 When Are Constructors or Destructors Called? 275
- 7.4.3 Friend Functions and Classes 275
- 7.4.4 Struct and Union 277
  - 7.4.4.1 Struct 277
  - 7.4.4.2 Union 278
- 7.4.5 Inline Functions 279
  - 7.4.5.1 Declaring Inline Functions 280
- 7.4.6 Static Members 280

7.5 Inheritance in C++ 281
- 7.5.1 Private Base Class 283
- 7.5.2 Extending a Class Declaration 283
- 7.5.3 Inheritance and Constructors/Destructors 284
- 7.5.4 Protected Members 285
- 7.5.5 Virtual Functions 285
- 7.5.6 Multiple Inheritance 287
  - 7.5.6.1 Construction/Destruction and Multiple Inheritance 288
  - 7.5.6.2 Virtual Base Class 288
  - 7.5.6.3 Resolving Ambiguity 289

7.6 Overloading/Overriding and Dynamic Binding in C++ 290
- 7.6.1 Function Overloading 290
- 7.6.2 Dynamic Binding 291
- 7.6.3 Operator Overloading 292

7.7 Templates 295
- 7.7.1 Templates in General 295
- 7.7.2 Template Functions 295
- 7.7.3 Template Classes 297

7.8 Streams in C++ 298
- 7.8.1 Streams in general 299
  - 7.8.1.1 Stream States 300
  - 7.8.1.2 Streams Class Definition 300
- 7.8.2 Extending Stream I/O Definition 301

7.9 Case Studies 303
- 7.9.1 Case Study One—Complex Numbers 303

7.9.1.1 Complex Numbers 303
7.9.1.2 Supporting a Complex Number System in C++ 304
7.9.2 Case Study Two—Electronic CAD 309
7.9.2.1 Computer-Aided Design 310
7.9.2.2 Design and Implementation of ECAD Components 311
7.9.2.3 An ECAD Sample Program 315
7.10 Summary 316

8

OBJECT-ORIENTED DATABASES 320

8.1 Introduction 320
8.1.1 What is an Object-Oriented Database? 321
8.1.2 Overview of DBMS Concepts 322
8.1.3 Approaches to Object-Oriented Databases 323
8.1.4 Organization of the Chapter 325
8.2 Foundation and Evolution of Object-Oriented Databases 325
8.2.1 Integrity Constraints of Object-Oriented Databases 329
8.3 Persistence in Object-Oriented Databases 331
8.3.1 Alternative Strategies for Persistence in Object-Oriented Databases 332
8.3.2 Defining and Manipulating Persistent Object-Oriented Databases 334
8.3.3 Extending a Database Language with Object-Oriented Capabilities 335
8.3.3.1 Case Study: Intelligent SQL 336
8.3.3.2 Case Study: UniSQL 340
8.3.4 Extending An Existing Object-Oriented Programming Language with Database Capabilities 341
8.3.4.1 Case Study: OPAL: Making Smalltalk a Database Language 342
8.3.4.2 C++ Classes for Database Management Capabilities 343
8.3.4.3 Case Study: Versant 346
8.3.4.4 Case Study: ObjectStore 346
8.3.5 Persistent Complex Object Storage Strategies 347
8.3.5.1 Case Study: ObjectStore 348
8.3.5.2 Case Study: GemStone 349
8.4 Transactions and Concurrency Control in Object-Oriented DBMSs 350
8.4.1 Long Transactions 352
8.4.1.1 Nested Transactions 353
8.4.1.2 Cooperating Transactions 353
8.4.2 Concurrency Control 354
8.4.2.1 Locking 354
8.4.2.2 Lock Modes 356