

AN INTRODUCTION TO
**Object-Oriented
Programming**

third edition



TIMOTHY BUDD

Preface

v

1 ■ Thinking Object-Oriented

1

- 1.1 Why Is OOP Popular? 2
- 1.2 Language and Thought 2
 - 1.2.1 *Eskimos and snow* 3
 - 1.2.2 *An example from computer languages* 3
 - 1.2.3 *Church's conjecture and the Whorf hypothesis* 5
- 1.3 A New Paradigm 7
- 1.4 A Way of Viewing the World 8
 - 1.4.1 *Agents and communities* 9
 - 1.4.2 *Messages and methods* 10
 - 1.4.3 *Responsibilities* 11
 - 1.4.4 *Classes and instances* 11
 - 1.4.5 *Class hierarchies—inheritance* 12
 - 1.4.6 *Method binding and overriding* 14
 - 1.4.7 *Summary of object-oriented concepts* 14
- 1.5 Computation as Simulation 15
 - 1.5.1 *The power of metaphor* 16
 - 1.5.2 *Avoiding infinite regression* 17
- 1.6 A Brief History 18
 - Summary 19
 - Further Reading 20
 - Self-Study Questions 22
 - Exercises 23

2 □ Abstraction 25

- 2.1 Layers of Abstraction 26
- 2.2 Other Forms of Abstraction 30
 - 2.2.1 *Division into parts* 32
 - 2.2.2 *Encapsulation and interchangeability* 32
 - 2.2.3 *Interface and implementation* 33
 - 2.2.4 *The service view* 34
 - 2.2.5 *Composition* 34
 - 2.2.6 *Layers of specialization* 36
 - 2.2.7 *Patterns* 38
- 2.3 A Short History of Abstraction Mechanisms 39
 - 2.3.1 *Assembly language* 39
 - 2.3.2 *Procedures* 40
 - 2.3.3 *Modules* 41
 - 2.3.4 *Abstract data types* 43
 - 2.3.5 *A service-centered view* 44
 - 2.3.6 *Messages, inheritance, and polymorphism* 44
- Summary 45
- Further Information 46
- Self-Study Questions 47
- Exercises 47

3 □ Object-Oriented Design 49

- 3.1 Responsibility Implies Noninterference 50
- 3.2 Programming in the Small and in the Large 51
- 3.3 Why Begin with Behavior? 51
- 3.4 A Case Study in RDD 52
 - 3.4.1 *The Interactive Intelligent Kitchen Helper* 53
 - 3.4.2 *Working through scenarios* 53
 - 3.4.3 *Identification of components* 54
- 3.5 CRC Cards—Recording Responsibility 55
 - 3.5.1 *Give components a physical representation* 55
 - 3.5.2 *The what/who cycle* 56
 - 3.5.3 *Documentation* 56
- 3.6 Components and Behavior 57

3.6.1	<i>Postponing decisions</i>	58
3.6.2	<i>Preparing for change</i>	59
3.6.3	<i>Continuing the scenario</i>	59
3.6.4	<i>Interaction diagrams</i>	61
3.7	Software Components	62
3.7.1	<i>Behavior and state</i>	62
3.7.2	<i>Instances and classes</i>	63
3.7.3	<i>Coupling and cohesion</i>	63
3.7.4	<i>Interface and implementation—Parnas's principles</i>	64
3.8	Formalize the Interface	65
3.8.1	<i>Coming up with names</i>	65
3.9	Designing the Representation	67
3.10	Implementing Components	67
3.11	Integration of Components	68
3.12	Maintenance and Evolution	69
	Summary	69
	Further Reading	70
	Self-Study Questions	70
	Exercises	71

4 ▣ Classes and Methods

73

4.1	Encapsulation	73
4.2	Class Definitions	74
4.2.1	<i>C++, Java, and C#</i>	75
4.2.2	<i>Apple Object Pascal and Delphi Pascal</i>	77
4.2.3	<i>Smalltalk</i>	77
4.2.4	<i>Other languages</i>	79
4.3	Methods	80
4.3.1	<i>Order of methods in a class declaration</i>	82
4.3.2	<i>Constant or immutable data fields</i>	83
4.3.3	<i>Separating definition and implementation</i>	83
4.4	Variations on Class Themes	87
4.4.1	<i>Methods without classes in Oberon</i>	87
4.4.2	<i>Interfaces</i>	88
4.4.3	<i>Properties</i>	89

4.4.4	<i>Forward definitions</i>	90
4.4.5	<i>Inner or nested classes</i>	91
4.4.6	<i>Class data fields</i>	94
4.4.7	<i>Classes as objects</i>	96
	Summary	97
	Further Reading	97
	Self-Study Questions	98
	Exercises	98

5 □ Messages, Instances, and Initialization 101

5.1	Message-Passing Syntax	101
5.2	Statically and Dynamically Typed Languages	103
5.3	Accessing the Receiver from within a Method	104
5.4	Object Creation	106
5.4.1	<i>Creation of arrays of objects</i>	107
5.5	Pointers and Memory Allocation	108
5.5.1	<i>Memory recovery</i>	109
5.6	Constructors	111
5.6.1	<i>The orthodox canonical class form</i>	115
5.6.2	<i>Constant values</i>	116
5.7	Destructors and Finalizers	117
5.8	Metaclasses in Smalltalk	120
	Summary	122
	Further Reading	122
	Self-Study Questions	123
	Exercises	123

6 □ A Case Study: The Eight-Queens Puzzle 125

6.1	The Eight-Queens Puzzle	125
6.1.1	<i>Creating objects that find their own solution</i>	126
6.2	Using Generators	127
6.2.1	<i>Initialization</i>	128
6.2.2	<i>Finding a solution</i>	129
6.2.3	<i>Advancing to the next position</i>	129

6.3	The Eight-Queens Puzzle in Several Languages	130
6.3.1	<i>The eight-queens puzzle in Object Pascal</i>	130
6.3.2	<i>The eight-queens puzzle in C++</i>	133
6.3.3	<i>The eight-queens puzzle in Java</i>	136
6.3.4	<i>The eight-queens puzzle in Objective-C</i>	139
6.3.5	<i>The eight-queens puzzle in Smalltalk</i>	142
6.3.6	<i>The eight-queens puzzle in Ruby</i>	143
	Summary	145
	Further Reading	145
	Self-Study Questions	145
	Exercises	146

7 **A Case Study: A Billiards Game** **147**

7.1	The Elements of Billiards	147
7.2	Graphical Objects	148
7.2.1	<i>The wall graphical object</i>	149
7.2.2	<i>The hole graphical object</i>	150
7.2.3	<i>The ball graphical object</i>	151
7.3	The Main Program	155
7.4	Using Inheritance	156
	Summary	159
	Further Information	159
	Self-Study Questions	159
	Exercises	160

8 **Inheritance and Substitution** **161**

8.1	An Intuitive Description of Inheritance	161
8.1.1	<i>The is-a test</i>	162
8.1.2	<i>Reasons to use inheritance</i>	162
8.2	Inheritance in Various Languages	164
8.3	Subclass, Subtype, and Substitution	166
8.3.1	<i>Substitution and strong typing</i>	167
8.4	Overriding and Virtual Methods	168
8.5	Interfaces and Abstract Classes	170

8.6	Forms of Inheritance	171
8.6.1	<i>Subclassing for specialization (subtyping)</i>	171
8.6.2	<i>Subclassing for specification</i>	171
8.6.3	<i>Subclassing for construction</i>	172
8.6.4	<i>Subclassing for generalization</i>	173
8.6.5	<i>Subclassing for extension</i>	174
8.6.6	<i>Subclassing for limitation</i>	174
8.6.7	<i>Subclassing for variance</i>	174
8.6.8	<i>Subclassing for combination</i>	175
8.6.9	<i>Summary of the forms of inheritance</i>	175
8.7	Variations on Inheritance	176
8.7.1	<i>Anonymous classes in Java</i>	176
8.7.2	<i>Inheritance and constructors</i>	177
8.7.3	<i>Virtual destructors</i>	178
8.8	The Benefits of Inheritance	179
8.8.1	<i>Software reusability</i>	179
8.8.2	<i>Code sharing</i>	179
8.8.3	<i>Consistency of interface</i>	179
8.8.4	<i>Software components</i>	179
8.8.5	<i>Rapid prototyping</i>	180
8.8.6	<i>Polymorphism and frameworks</i>	180
8.8.7	<i>Information hiding</i>	180
8.9	The Costs of Inheritance	181
8.9.1	<i>Execution speed</i>	181
8.9.2	<i>Program size</i>	181
8.9.3	<i>Message-passing overhead</i>	181
8.9.4	<i>Program complexity</i>	182
	Summary	182
	Further Reading	183
	Self-Study Questions	183
	Exercises	184

9 □ A Case Study—A Card Game

187

9.1	The Class <code>PlayingCard</code>	187
9.2	Data and View Classes	189
9.3	The Game	190

9.4	Card Piles—Inheritance in Action	191
9.4.1	<i>The default card pile</i>	193
9.4.2	<i>The suit piles</i>	194
9.4.3	<i>The deck pile</i>	194
9.4.4	<i>The discard pile</i>	196
9.4.5	<i>The tableau piles</i>	197
9.5	Playing the Polymorphic Game	199
9.6	The Graphical User Interface	199
	Summary	204
	Further Reading	204
	Self-Study Questions	204
	Exercises	205

10 □ Subclasses and Subtypes 207

10.1	Substitutability	207
10.2	Subtypes	208
10.3	The Substitutability Paradox	211
10.3.1	<i>Is this a problem?</i>	212
10.4	Subclassing for Construction	212
10.4.1	<i>Private inheritance in C++</i>	214
10.5	Dynamically Typed Languages	215
10.6	Pre- and Postconditions	216
10.7	Refinement Semantics	217
	Summary	218
	Further Reading	218
	Self-Study Questions	219
	Exercises	219

11 □ Static and Dynamic Behavior 221

11.1	Static versus Dynamic Typing	221
11.2	Static and Dynamic Classes	223
11.2.1	<i>Run-time type determination</i>	225
11.2.2	<i>Down casting (reverse polymorphism)</i>	227

11.2.3	<i>Run-time testing without language support</i>	227
11.2.4	<i>Testing message understanding</i>	229
11.3	Static versus Dynamic Method Binding	230
	Summary	232
	Further Reading	233
	Self-Study Questions	233
	Exercises	234

12 □ Implications of Substitution 235

12.1	Memory Layout	235
12.1.1	<i>Minimum static space allocation</i>	237
12.1.2	<i>Maximum static space allocation</i>	240
12.1.3	<i>Dynamic memory allocation</i>	240
12.2	Assignment	242
12.2.1	<i>Assignment in C++</i>	242
12.3	Copies and Clones	245
12.3.1	<i>Copies in Smalltalk and Objective-C</i>	245
12.3.2	<i>Copy constructors in C++</i>	245
12.3.3	<i>Cloning in Java</i>	246
12.4	Equality	247
12.4.1	<i>Equality and identity</i>	247
12.4.2	<i>The paradoxes of equality testing</i>	248
	Summary	250
	Further Reading	251
	Self-Study Questions	251
	Exercises	251

13 □ Multiple Inheritance 253

13.1	Inheritance as Categorization	254
13.1.1	<i>Incomparable complex numbers</i>	255
13.2	Problems Arising from Multiple Inheritance	257
13.2.1	<i>Name ambiguity</i>	257
13.2.2	<i>Impact on substitutability</i>	259
13.2.3	<i>Redefinition in Eiffel</i>	260
13.2.4	<i>Resolution by class ordering in CLOS</i>	261

13.3	Multiple Inheritance of Interfaces	263
13.3.1	<i>Mixins in CLOS</i>	266
13.4	Inheritance from Common Ancestors	267
13.4.1	<i>Constructors and multiple inheritance</i>	270
13.5	Inner Classes	271
	Summary	272
	Further Reading	273
	Self-Study Questions	273
	Exercises	273

14 ▣ Polymorphism and Software Reuse 275

14.1	Polymorphism in Programming Languages	275
14.1.1	<i>Many tools, one goal</i>	277
14.2	Mechanisms for Software Reuse	277
14.2.1	<i>Using composition</i>	278
14.2.2	<i>Using inheritance</i>	280
14.2.3	<i>Composition and inheritance contrasted</i>	282
14.3	Efficiency and Polymorphism	283
14.4	Will Widespread Software Reuse Become Reality?	284
	Summary	285
	Further Information	285
	Self-Study Questions	286
	Exercises	286

15 ▣ Overloading 287

15.1	Type Signatures and Scopes	288
15.2	Overloading Based on Scopes	289
15.3	Overloading Based on Type Signatures	290
15.3.1	<i>Coercion and conversion</i>	293
15.4	Redefinition	299
15.5	Polyadicity	301
15.5.1	<i>Optional parameters</i>	303
15.6	Multi-Methods	303

15.6.1	<i>Overloading Based on Values</i>	306
	Summary	306
	Further Information	306
	Self-Study Questions	307
	Exercises	307

16 □ Overriding 309

16.1	Notating Overriding	311
16.2	Replacement versus Refinement	313
16.2.1	<i>Replacement in Smalltalk</i>	313
16.2.2	<i>Refinement in Beta</i>	316
16.2.3	<i>Refinement and the subclass/subtype distinction</i>	319
16.2.4	<i>Wrappers in CLOS</i>	319
16.3	Deferred Methods	320
16.4	Overriding versus Shadowing	322
16.5	Covariance and Contravariance	324
16.6	Variations on Overriding	329
16.6.1	<i>Final methods in Java</i>	329
16.6.2	<i>Versioning in C#</i>	330
	Summary	331
	Further Information	332
	Self-Study Questions	332
	Exercises	332

17 □ The Polymorphic Variable 335

17.1	Simple Polymorphic Variables	335
17.2	The Receiver Variable	337
17.2.1	<i>The role of the polymorphic variable in frameworks</i>	339
17.2.2	<i>Endpoint comparisons in Smalltalk</i>	340
17.2.3	<i>Self and super</i>	341
17.3	Downcasting	343
17.4	Pure Polymorphism	344
	Summary	347

Further Information	347
Self-Study Questions	348
Exercises	348

18 ▣ Generics 349

18.1	Template Functions	349
18.2	Template Classes	351
18.2.1	<i>Bounded genericity</i>	353
18.3	Inheritance in Template Arguments	353
18.3.1	<i>Inheritance and arrays</i>	355
18.4	Case Study—Combining Separate Classes	356
	Summary	360
	Further Reading	360
	Self-Study Questions	360
	Exercises	360

19 ▣ Container Classes 363

19.1	Containers in Dynamically Typed Languages	363
19.1.1	<i>Containers in Smalltalk-80</i>	364
19.2	Containers in Statically Typed Languages	365
19.2.1	<i>The tension between typing and reuse</i>	366
19.2.2	<i>Substitution and downcasting</i>	368
19.2.3	<i>Using substitution and overriding</i>	372
19.2.4	<i>Parameterized classes</i>	374
19.3	Restricting Element Types	376
19.4	Element Traversal	378
19.4.1	<i>Iterator loops</i>	379
19.4.2	<i>The visitor approach</i>	381
	Summary	385
	Further Reading	386
	Self-Study Questions	387
	Exercises	387

20 □ A Case Study: The STL	389
20.1 Iterators	391
20.2 Function Objects	393
20.3 Example Program—An Inventory System	394
20.4 Example Program—Graphs	396
20.4.1 <i>Shortest path algorithm</i>	399
20.4.2 <i>Developing the data structures</i>	399
20.5 A Concordance	402
20.6 The Future of OOP	405
Summary	406
Further Reading	406
Self-Study Questions	406
Exercises	406
21 □ Frameworks	407
21.1 Reuse and Specialization	407
21.1.1 <i>High- and low-level abstractions</i>	410
21.1.2 <i>An upside-down library</i>	412
21.2 Example Frameworks	413
21.2.1 <i>The Java Applet API</i>	413
21.2.2 <i>A Simulation Framework</i>	414
21.2.3 <i>An event-driven simulation framework</i>	415
Summary	422
Further Reading	422
Self-Study Questions	422
Exercises	422
22 □ An Example Framework: The AWT and Swing	423
22.1 The AWT Class Hierarchy	423
22.2 The Layout Manager	426
22.3 Listeners	428
22.3.1 <i>Adapter classes</i>	429
22.4 User Interface Components	430

22.5	Case Study: A Color Display	433
22.6	The Swing Component Library	437
22.6.1	<i>Import libraries</i>	437
22.6.2	<i>Different components</i>	437
22.6.3	<i>Different paint protocol</i>	438
22.6.4	<i>Adding components to a window</i>	438
	Summary	438
	Further Reading	439
	Self-Study Questions	439
	Exercises	439

23 □ Object Interconnections

441

23.1	Coupling and Cohesion	442
23.1.1	<i>Varieties of coupling</i>	442
23.1.2	<i>Varieties of cohesion</i>	445
23.1.3	<i>The Law of Demeter</i>	447
23.1.4	<i>Class-level versus object-level visibility</i>	448
23.1.5	<i>Active values</i>	449
23.2	Subclass Clients and User Clients	450
23.3	Control of Access and Visibility	451
23.3.1	<i>Visibility in Smalltalk</i>	451
23.3.2	<i>Visibility in Object Pascal</i>	452
23.3.3	<i>Visibility in C++</i>	452
23.3.4	<i>Visibility in Java</i>	457
23.3.5	<i>Visibility in Objective-C</i>	458
23.4	Intentional Dependency	459
	Summary	460
	Further Reading	460
	Self-Study Questions	461
	Exercises	461

24 □ Design Patterns

463

24.1	Controlling Information Flow	464
24.2	Describing Patterns	465
24.3	Iterator	466

24.4	Software Factory	467
24.5	Strategy	468
24.6	Singleton	469
24.7	Composite	469
24.8	Decorator	471
24.9	The Double-Dispatching Pattern	472
24.10	Flyweight	474
24.11	Proxy	474
24.12	Facade	475
24.13	Observer	475
	Summary	476
	Further Reading	477
	Self-Study Questions	478
	Exercises	478

25 □ Reflection and Introspection

479

25.1	Mechanisms for Understanding	479
25.1.1	<i>Class objects</i>	479
25.1.2	<i>The class name as string</i>	481
25.1.3	<i>Testing the class of an object</i>	481
25.1.4	<i>Creating an instance from a class</i>	483
25.1.5	<i>Testing if an object understands a message</i>	484
25.1.6	<i>Class behavior</i>	484
25.2	Methods as Objects	485
25.3	Mechanisms for Modification	486
25.3.1	<i>Method editing in Smalltalk</i>	486
25.3.2	<i>Dynamic class loading in Java</i>	487
25.4	Metaclasses	489
	Summary	491
	Further Reading	491
	Self-Study Questions	492

26	▣ Distributed Objects	493
26.1	Addresses, Ports, and Sockets	494
26.2	A Simple Client/Server Program	496
26.3	Multiple Clients	498
26.4	Transmitting Objects over a Network	504
26.5	Providing More Complexity	507
	Summary	507
	Further Reading	508
	Self-Study Questions	508
	Exercises	508
27	▣ Implementation	511
27.1	Compilers and Interpreters	511
27.2	The Receiver as Argument	512
27.3	Inherited Methods	513
27.3.1	<i>The problem of multiple inheritance</i>	514
27.3.2	<i>The slicing problem</i>	515
27.4	Overridden Methods	515
27.4.1	<i>Eliminating virtual calls and in-lining</i>	517
27.5	Name Encoding	518
27.6	Dispatch Tables	518
27.6.1	<i>A method cache</i>	520
27.7	Bytecode Interpreters	521
27.8	Just-in-Time Compilation	523
	Summary	524
	Further Reading	524
	Self-Study Questions	525
	Exercises	525
A	▣ Source for the Eight-Queens Puzzle	527
A.1	Eight-Queens in Apple Object Pascal	527
A.2	Eight-Queens in C++	530

A.3	Eight-Queens in Java	532
A.4	Eight-Queens in Objective-C	536
A.5	Eight-Queens in Ruby	539
A.6	Eight-Queens in Smalltalk	541

B □	Source for the Billiards Game	543
------------	--------------------------------------	------------

B.1	The Version without Inheritance	543
B.2	The Version with Inheritance	552

C □	Source for the Solitaire Game	557
------------	--------------------------------------	------------

<i>Glossary</i>	569
<i>References</i>	585
<i>Index</i>	599