

Jesse Feiler



In **Full Color**

**SECOND  
EDITION**

Figures and  
code appear  
as they do  
in Xcode 5

Sams **Teach Yourself**

# Objective-C

in **24**  
**Hours**

**SAMS**

FREE SAMPLE CHAPTER

SHARE WITH OTHERS



Jesse Feiler

Sams **Teach Yourself**  
**Objective-C**

in **24**  
**Hours**

Second Edition

**SAMS**

800 East 96th Street, Indianapolis, Indiana, 46240 USA

## Sams Teach Yourself Objective-C in 24 Hours, Second Edition

Copyright © 2014 by Pearson Education, Inc.

All rights reserved. No part of this book shall be reproduced, stored in a retrieval system, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise, without written permission from the publisher. No patent liability is assumed with respect to the use of the information contained herein. Although every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions. Nor is any liability assumed for damages resulting from the use of the information contained herein.

ISBN-13: 978-0-672-33449-8

ISBN-10: 0-672-33449-6

Library of Congress Control Number: 2013954664

Printed in the United States of America

First Printing March 2014

### Trademarks

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Sams Publishing cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

### Warning and Disclaimer

Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an “as is” basis. The author and the publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book.

### Special Sales

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at [corpsales@pearsoned.com](mailto:corpsales@pearsoned.com) or (800) 382-3419.

For government sales inquiries, please contact [governmentsales@pearsoned.com](mailto:governmentsales@pearsoned.com).

For questions about sales outside the U.S., please contact [international@pearsoned.com](mailto:international@pearsoned.com).

### Editor-in-Chief

Greg Wiegand

### Executive Editor

Loretta Yates

### Development Editor

Sondra Scott

### Marketing Manager

Stephane Nakib

### Managing Editor

Kristy Hart

### Senior Project Editor

Lori Lyons

### Copy Editor

Karen Annett

### Indexer

Erika Millen

### Proofreader

Kathy Ruiz

### Technical Editor

Robert McGovern

### Publishing Coordinator

Cindy Teeters

### Cover Designer

Mark Shirar

### Compositor

Nonie Ratcliff

### Manufacturing Buyer

Dan Uhrig

# Contents at a Glance

Introduction .....	1
--------------------	---

## Part I: Getting Started with Objective-C

<b>HOUR 1</b> Overview of the Developer Program .....	7
<b>2</b> Object-Oriented Programming with Objective-C .....	21
<b>3</b> Using Object-Oriented Features in Objective-C .....	33
<b>4</b> Using Xcode 5 .....	41
<b>5</b> Using Compiler Directives .....	63

## Part II: Working with the Objective-C Basics

<b>HOUR 6</b> Exploring Messaging and a Testbed App .....	77
<b>7</b> Declaring a Class in an Interface File .....	97
<b>8</b> Declaring Instance Variables in an Interface File .....	111
<b>9</b> Declaring Properties in an Interface File .....	127
<b>10</b> Declaring Methods in an Interface File .....	141
<b>11</b> Declaring Actions in an Interface File .....	149
<b>12</b> Routing Messages with Selectors .....	165
<b>13</b> Building on the Foundation .....	179
<b>14</b> Defining a Class in an Implementation File .....	189
<b>15</b> Organizing Data with Collections .....	205
<b>16</b> Managing Memory and Runtime Objects .....	221

## Part III: Expanding and Extending Classes

<b>17</b> Extending a Class with Protocols and Delegates .....	231
<b>18</b> Extending a Class with Categories and Extensions .....	239
<b>19</b> Using Associative References and Fast Enumeration .....	249
<b>20</b> Working with Blocks .....	259

**Part IV: Beyond the Basics**

**21** Handling Exceptions and Errors ..... 271  
**22** Grand Central Dispatch: Using Queues and Threading ..... 283  
**23** Working with the Debugger ..... 293  
**24** Using Xcode Debug Gauges for Analysis ..... 305

**Part V: Appendixes**

**A** C Syntax Summary ..... 313  
**B** Apps, Packages, and Bundles ..... 317  
**C** Archiving and Packaging Apps for Development and Testing ..... 321  
Index ..... 325

# Table of Contents

<b>Introduction</b>	<b>1</b>
Who Should Read This Book .....	1
What This Book Covers .....	1
Downloading the Sample Files .....	2
How This Book Is Organized .....	2
Part I: Getting Started with Objective-C .....	2
Part II: Working with the Objective-C Basics .....	3
Part III: Expanding and Extending Classes .....	4
Part IV: Beyond the Basics .....	4
Part V: Appendixes .....	5
<b>Part I: Getting Started with Objective-C</b>	
<b>HOURL 1: Overview of the Developer Program</b>	<b>7</b>
Introducing Objective-C .....	7
Enrolling as an Apple Developer .....	8
Choosing Your Program .....	8
Selecting Your Membership Category .....	9
Registering for Your Apple ID .....	10
Setting Up the Development Environment .....	11
Talking Xcode .....	11
Introducing the Xcode Workspace .....	12
Getting Up and Running with Your First Project .....	16
Summary .....	17
Q&A .....	18
Workshop .....	18
Quiz .....	18
Quiz Answers .....	18
Activities .....	19

<b>HOURL 2: Object-Oriented Programming with Objective-C</b>	<b>21</b>
Object-Oriented Programming in the Objective-C World .....	21
Implementing Object-Oriented Programming .....	21
Building Object-Oriented Projects .....	22
Looking at the Frameworks .....	23
Creating C with Objects .....	24
Start Thinking in Objective-C .....	24
Understanding Data Abstraction .....	24
Considering Classes and Instances .....	25
Exploring Encapsulation .....	25
Using Accessors to Manage Encapsulation .....	28
Managing Inheritance in the Objective-C World .....	29
Summary .....	30
Q&A .....	31
Workshop .....	31
Quiz .....	31
Quiz Answers .....	31
Activities .....	31
<b>HOURL 3: Using Object-Oriented Features in Objective-C</b>	<b>33</b>
Communicating to Methods with Messages .....	33
Looking at a Simple Message .....	33
Declaring a Method .....	34
Using Class Methods .....	35
Working with <code>id</code> —Strongly and Weakly Typed Variables .....	35
Nesting Messages .....	36
Looking at Method Signatures and Parameters .....	36
Allocating and Initializing Objects .....	37
Summary .....	39
Q&A .....	39
Workshop .....	39
Quiz .....	39
Quiz Answers .....	39
Activities .....	39

<b>HOUR 4: Using Xcode 5</b>	<b>41</b>
Getting to Work with Xcode .....	41
Keeping Track of Your Source Code .....	46
Exploring Source Code Control .....	47
Working in a Source Code Repository World .....	48
Using Git with Xcode .....	50
Using a Remote Repository .....	58
Summary .....	61
Q&A .....	61
Workshop .....	61
Quiz .....	61
Quiz Answers .....	61
Activities .....	62
<b>HOUR 5: Using Compiler Directives</b>	<b>63</b>
Exploring Your Projects .....	63
Looking at the iOS Project .....	64
Looking at the OS X Project .....	64
Looking at Both Projects .....	65
Working with Compiler Directives .....	66
Working with Basic Directives .....	66
Looking at Prefix Headers .....	67
Looking at Plain C Code in <code>main.m</code> .....	69
Investigating Header ( <code>.h</code> ) Files .....	70
Looking Inside Message ( <code>.m</code> ) Files .....	71
Using Objective-C Compiler Directives .....	74
Summary .....	74
Q&A .....	74
Workshop .....	75
Quiz .....	75
Quiz Answers .....	75
Activities .....	75



## Part II: Working with the Objective-C Basics

<b>HOUR 6: Exploring Messaging and a Testbed App</b>	<b>77</b>
Setting Up the Testbed Apps	77
Adding a Text Field and Connecting It to Your Code	81
Adding the Text Field	82
Connecting the Text Field to the Code	88
Sending a Message to the Text Field	92
Reviewing the Message Syntax	94
Summary	95
Q&A	95
Workshop	96
Quiz	96
Quiz Answers	96
Activities	96
<b>HOUR 7: Declaring a Class in an Interface File</b>	<b>97</b>
Letting Xcode Do the Work	97
Designing Classes	97
Getting Ready to Create the Class	98
Exploring Class Hierarchies	104
Calling Superclass Methods	104
Introducing Protocols	106
Declaring Classes	106
Writing a Basic Class Declaration	106
Using Forward References	107
Summary	109
Q&A	109
Workshop	109
Quiz	109
Quiz Answers	109
Activities	109

<b>HOUR 8: Declaring Instance Variables in an Interface File</b>	<b>111</b>
Declaring Instance Variables and Properties .....	111
Using the Class .....	111
Placing a Class Instance in Context .....	112
Choosing the Context .....	113
Creating an Instance Variable for <code>CurrencyConverter</code> with <code>id</code> .....	114
What Happens When Execution Stops .....	115
Dynamic Binding .....	117
Creating an Instance Variable for <code>CurrencyConverter</code> with the Class Name .....	117
Creating an Instance Variable for <code>CurrencyConverter</code> with a Superclass Name .....	119
Managing Instance Variable Visibility .....	123
Summary .....	124
Q&A .....	124
Workshop .....	124
Quiz .....	124
Quiz Answers .....	124
Activities .....	125
<b>HOUR 9: Declaring Properties in an Interface File</b>	<b>127</b>
Comparing Interface Variables and Properties .....	127
Reviewing Basic Variable Declarations .....	128
Creating Declared Properties: The Basics .....	130
Dealing with Memory for Objects .....	132
Working with Attributes for Declared Properties .....	133
Using Declared Properties .....	133
Accessing the Property with Message Syntax .....	134
Accessing the Property with Dot Syntax .....	134
Using Attributes .....	135
Accessor Methods .....	136
Writability .....	136

Setter Semantics .....	137
Atomicity .....	137
Using Other Attribute Decorators .....	137
Implementing Properties .....	138
Creating Accessors with @synthesize .....	138
Promising Data with @dynamic .....	139
Summary .....	139
Q&A .....	139
Workshop .....	139
Quiz .....	139
Quiz Answers .....	139
Activities .....	140
<b>HOOR 10: Declaring Methods in an Interface File</b> .....	<b>141</b>
Working with Methods in a Class .....	141
Reviewing Method Syntax .....	142
Distinguishing Between Class and Instance Methods .....	142
Exploring the Method Declaration .....	143
Writing the Method Declaration .....	146
Returning Complex Data Structures from Methods .....	146
Summary .....	147
Q&A .....	147
Workshop .....	148
Quiz .....	148
Quiz Answers .....	148
Activities .....	148
<b>HOOR 11: Declaring Actions in an Interface File</b> .....	<b>149</b>
Introducing Actions .....	149
What Actions Can Do for You .....	150
Comparing Actions in OS X and iOS .....	159
Disconnecting Actions .....	161
Summary .....	162
Q&A .....	162

Workshop .....	163
Quiz .....	163
Quiz Answers .....	163
Activities .....	163
<b>HOOR 12: Routing Messages with Selectors</b> .....	<b>165</b>
Getting Inside Objective-C Messages .....	165
Receiver and Selector Objects in Messages .....	166
Getting Inside the Objective-C Runtime .....	167
Working with SEL and @selector () .....	168
Using performSelector .....	169
Creating a Selector with @selector () .....	169
Creating a Selector from a String .....	169
Using a Selector .....	170
Using NSInvocation .....	172
Creating an NSInvocation .....	172
Using NSInvocation Properties .....	173
Invoking an NSInvocation .....	175
Testing Whether an Instance Can Respond to a Selector .....	175
Summary .....	176
Q&A .....	176
Workshop .....	177
Quiz .....	177
Quiz Answers .....	177
Activities .....	177
<b>HOOR 13: Building on the Foundation</b> .....	<b>179</b>
Exploring the Foundation Framework .....	179
Foundation Classes .....	180
Root Classes .....	180
Other Classes .....	181
Foundation Paradigms and Policies .....	182
Mutability .....	182
Class Clusters .....	183
Notifications .....	184

Summary .....	187
Q&A .....	187
Workshop .....	187
Quiz .....	187
Quiz Answers .....	187
Activities .....	187
<b>HOOR 14: Defining a Class in an Implementation File</b> .....	<b>189</b>
Working with a New Project .....	189
Reconsidering Dynamic Typing .....	190
Designing the (Currency) Converter .....	190
Creating a New App .....	193
Implementing a Method .....	197
Expanding the Class with <code>init</code> Methods .....	200
Summary .....	202
Q&A .....	202
Workshop .....	202
Quiz .....	202
Quiz Answers .....	202
Activities .....	203
<b>HOOR 15: Organizing Data with Collections</b> .....	<b>205</b>
Collecting Objects .....	205
Getting Familiar with Property Lists .....	207
Using Collections in Property Lists .....	208
Building Collections from Property Lists at Runtime .....	209
Comparing the Collection Classes .....	209
Creating a Collection .....	210
Using the Common Collection Creation Methods .....	211
Using Objective-C Literal Syntax .....	212
Reading and Writing Arrays .....	213
Reading and Writing Dictionaries .....	213
Creating Sets .....	214

Enumerating a Collection .....	214
Examining <code>NSEnumerator</code> Methods .....	214
Creating <code>NSEnumerator</code> Instances for Collections .....	215
Testing Membership in a Collection .....	217
Accessing an Object in a Collection .....	218
Summary .....	218
Q&A .....	218
Workshop .....	219
Quiz .....	219
Quiz Answers .....	219
Activities .....	219
<b>HOUR 16: Managing Memory and Runtime Objects</b> .....	<b>221</b>
Managing Objects in Memory .....	221
Managing Reference Counts Manually .....	222
Looking at Memory Management Before ARC .....	223
Summarizing Memory Management .....	225
Managing Reference Counts with ARC .....	225
Using Declared Property Attributes .....	226
Variable Qualifiers .....	227
Autoreleasing Variables .....	228
Summary .....	229
Q&A .....	229
Workshop .....	229
Quiz .....	229
Quiz Answers .....	229
Activities .....	229
<b>Part III: Expanding and Extending Classes</b>	
<b>HOUR 17: Extending a Class with Protocols and Delegates</b> .....	<b>231</b>
Exploring the Pros and Cons of Subclassing .....	231
Introducing the Example .....	232
Working with Protocols .....	232
Working with Delegates .....	233

- Putting Protocols and Delegates Together ..... 233
- Looking Deeper Inside Protocols ..... 236
- Summary ..... 236
- Q&A ..... 236
- Workshop ..... 237
  - Quiz ..... 237
  - Quiz Answers ..... 237
  - Activities ..... 237

**HOUR 18: Extending a Class with Categories and Extensions 239**

- Comparing Categories and Protocols ..... 239
  - Choosing When to Use a Category ..... 240
  - Comparing Other Techniques with Categories ..... 240
- Comparing Categories with Subclasses ..... 241
  - Modifying a Class Hierarchy ..... 241
  - Confining Modifications to Categories ..... 242
- Working with Categories ..... 242
- Using Class Extensions ..... 245
- Working with Informal Protocols ..... 246
- Summary ..... 246
- Q&A ..... 246
- Workshop ..... 247
  - Quiz ..... 247
  - Quiz Answers ..... 247
  - Activities ..... 247

**HOUR 19: Using Associative References and Fast Enumeration 249**

- Catching Up on Objective-C 2.0 Time-Saving Features ..... 249
- Extending Classes by Adding Instance Variables (Sort of) ..... 250
  - Adding an Associative Reference ..... 251
  - Getting and Setting an Associative Reference ..... 252
  - Removing an Associative Reference for a Key ..... 254
  - Removing All Associative References from an Object ..... 254

Using Fast Enumeration .....	254
Using Fast Enumeration .....	254
Using Fast Enumeration with an NSEnumerator .....	256
Summary .....	256
Q&A .....	256
Workshop .....	257
Quiz .....	257
Quiz Answers .....	257
Activities .....	257
<b>HOOR 20: Working with Blocks</b> .....	<b>259</b>
Revisiting Blocks .....	259
Looking at Callbacks .....	260
Considering Callback Routines .....	263
Introducing Blocks .....	264
Creating a Block as a Block Variable .....	264
Using a Block Variable .....	265
Exploring Blocks in Cocoa .....	266
NSString enumerateLinesUsingBlock: .....	267
NSArray enumerateObjectsUsingBlock: .....	268
NSSet enumerateObjectsUsingBlock: .....	268
Looking Deeper into Cocoa Blocks and Memory .....	269
Summary .....	269
Q&A .....	269
Workshop .....	270
Quiz .....	270
Quiz Answers .....	270
Activities .....	270



**Part IV: Beyond the Basics**

**HOURL 21: Handling Exceptions and Errors** **271**

- Rethinking Exceptions and Errors ..... 271
- Introducing the Exception and Error Classes ..... 272
  - Using Exceptions ..... 273
  - Using Errors ..... 273
  - Looking Inside NSError ..... 274
  - Looking Inside NSError ..... 274
- Identifying an Exception ..... 277
- Throwing an Exception ..... 278
- Catching an Exception ..... 280
- Summary ..... 280
- Q&A ..... 280
- Workshop ..... 280
  - Quiz ..... 280
  - Quiz Answers ..... 281
  - Activities ..... 281

**HOURL 22: Grand Central Dispatch: Using Queues and Threading** **283**

- Getting Started with Concurrency ..... 283
  - Looking at Processors Inside Computers ..... 283
  - Using Concurrency Without Rewriting User Apps ..... 284
  - Using Threads for Concurrency ..... 284
  - Introducing Grand Central Dispatch (GCD) ..... 285
- Introducing Queues ..... 286
  - Dispatch Queues ..... 287
  - Dispatch Sources ..... 287
  - Operation Queues ..... 287
- Using Dispatch Queues ..... 288
  - Using Global Concurrent Dispatch Queues ..... 288
  - Adding a Task to a Global Concurrent Queue ..... 289
  - Designing with Queues ..... 290
- Summary ..... 290

Q&A .....	290
Workshop .....	291
Quiz .....	291
Quiz Answers .....	291
Activities .....	291
<b>HOOR 23: Working with the Debugger</b> .....	<b>293</b>
Logging Information .....	293
Using Console Logs .....	294
Using NSLog .....	295
Enhancing NSLog .....	295
Using Smart Breakpoints .....	297
Enhancing Breakpoints with Messages .....	298
Breaking on a Condition .....	300
Summary .....	302
Q&A .....	302
Workshop .....	303
Quiz .....	303
Quiz Answers .....	303
Activities .....	303
<b>HOOR 24: Using Xcode Debug Gauges for Analysis</b> .....	<b>305</b>
Putting Debug Gauges in Perspective .....	305
Monitoring CPU Utilization .....	306
Monitoring Memory Utilization .....	307
Monitoring Energy .....	308
Using Instruments .....	310
Summary .....	311
Q&A .....	311
Workshop .....	312
Quiz .....	312
Quiz Answers .....	312
Activities .....	312

**Part V: Appendixes**

<b>APPENDIX A: C Syntax Summary</b>	<b>313</b>
Data Types .....	313
Enumerated Type .....	313
Struct Type .....	314
Pointers .....	314
Arrays .....	315
Control Structures .....	315
if Statements .....	315
switch Statements .....	315
Repeat Statements .....	316
<b>APPENDIX B: Apps, Packages, and Bundles</b>	<b>317</b>
Looking Inside a Project Bundle .....	317
lproj Files .....	318
Asset Catalogs .....	318
plist Files .....	319
Precompiled Header Files (.pch) .....	319
<b>APPENDIX C: Archiving and Packaging Apps for Development and Testing</b>	<b>321</b>
Archiving .....	321
<b>Index</b>	<b>325</b>

# About the Author

**Jesse Feiler** is a developer and author. He has been an Apple developer since before it became fashionable, and has worked with mobile devices starting with Apple's Newton and continuing with the iOS products (iPhone, iPod touch, and iPad).

His books include *Sams Teach Yourself Core Data in 24 Hours*, *Data-Driven iOS Apps for iPad and iPhone with FileMaker Pro*, *FileMaker Pro in Depth* (Sams/Pearson), *Database-Driven Web Sites* (Harcourt), *Learning iCloud Data Management* (Addison Wesley/Pearson), and *iOS App Development for Dummies* (Wiley).

He has written about Objective-C and the Apple frameworks beginning with *Rhapsody Developer's Guide* (AP Professional) and *Mac OS X Developer's Guide* (Morgan Kaufmann).

He is the author of *MinutesMachine*, the meeting management software for iPad, as well as Saranac River Trail app for iPhone and iPad. There are more details at [champlainarts.com](http://champlainarts.com).

A native of Washington DC, he has lived in New York City and currently lives in Plattsburgh, NY. He can be reached at [northcountryconsulting.com](http://northcountryconsulting.com).

# Acknowledgments

As always, Carole Jelen at Waterside Productions has provided help and guidance in bringing this book to fruition. At Pearson, Loretta Yates has helped move this book from an idea to an actual book. Along the way, the book and author have benefited from technical suggestions from Robert McGovern. The production staff kept the book on track and helped clarify text.

# We Want to Hear from You!

As the reader of this book, you are our most important critic and commentator. We value your opinion and want to know what we're doing right, what we could do better, what areas you'd like to see us publish in, and any other words of wisdom you're willing to pass our way.

We welcome your comments. You can email or write to let us know what you did or didn't like about this book—as well as what we can do to make our books better.

*Please note that we cannot help you with technical problems related to the topic of this book.*

When you write, please be sure to include this book's title and author as well as your name and email address. We will carefully review your comments and share them with the author and editors who worked on the book.

Email: [consumer@sampublishing.com](mailto:consumer@sampublishing.com)

Mail: Sams Publishing  
ATTN: Reader Feedback  
800 East 96th Street  
Indianapolis, IN 46240 USA

## Reader Services

Visit our website and register this book at [informit.com/register](http://informit.com/register) for convenient access to any updates, downloads, and errata that might be available for this book.

# Introduction

If you want to develop native apps for OS X or for iOS, the language in which you write them is Objective-C. (You can write web-based apps using Safari extensions and HTML5.)

Because Apple's development environment builds on the frameworks of Cocoa and Cocoa Touch, what you have to write is the code that is specific to your own app. All the basic functionality is provided by Cocoa and the frameworks; you do not have to write any code to manage menus, for example (in most cases), and you don't have to worry about implementing complex view structures on iOS such as navigation bars and split views. The code that you write is very specific code that fits into the existing frameworks. It might be code that overrides or extends an existing method, or it might be new methods that are unique to your own app.

## Who Should Read This Book

Even though the code that you write might not be extensive, you do have to understand the syntax and structure of Objective-C in order to write it and to understand the code that Apple provides for you.

You should have some basic experience with programming in order to read this book, but it is designed to bring you up to speed on the major prerequisites to understanding Objective-C, including the concepts of object-oriented programming. The book assumes that you have at least a basic understanding of the C programming language on which Objective-C is based. Don't worry if you are far from a C expert; the concepts used in Objective-C are the basics.

## What This Book Covers

Objective-C has been the language of choice for the Cocoa environments and, before them, the NeXTSTEP and Rhapsody environments. Theoretically, it can be used in other contexts, but in practice, it is the language of these environments. Sometimes, drawing the line between Objective-C syntax and elements of the Cocoa environments and frameworks can be very difficult. This book focuses on the language itself. It provides a number of examples from code in

the Cocoa frameworks as well as code in the examples from [developer.apple.com](http://developer.apple.com) and the templates built in to Xcode, but the primary focus is on the language. The book includes information on how to use the Xcode development tool because, for all intents and purposes, you have to use it to write Objective-C code.

## Downloading the Sample Files

You can download examples from the author's website at [northcountryconsulting.com](http://northcountryconsulting.com) or from the publisher's site at [informit.com/title/9780672334498](http://informit.com/title/9780672334498).

## How This Book Is Organized

There are five parts to this book. You can focus on whichever one addresses an immediate problem, or you can get a good overview by reading straight through. As with all of the Sams Teach Yourself books, as much as possible each hour is made to stand on its own so that you can jump around to learn in your own way. Cross-references throughout the book help you find related material.

## Part I: Getting Started with Objective-C

This part gives you the high-level view of Objective-C.

- ▶ **Hour 1, “Overview of the Developer Program”—**Here you see how to register to become an Apple developer and gain access to the resources on [developer.apple.com](http://developer.apple.com). You can choose from various options, but you must register before you can do any meaningful development work.
- ▶ **Hour 2, “Object-Oriented Programming with Objective-C”—**This hour shows you how Objective-C builds on top of C. It also covers how Objective-C implements object-oriented concepts and manages inheritance.
- ▶ **Hour 3, “Using Object-Oriented Features in Objective-C”—**Objective-C is a dynamic language that relies on messaging. Those concepts are explained here so that you can begin to implement your own apps.
- ▶ **Hour 4, “Using Xcode 5”—** This is the integrated development environment (IDE) you use to develop apps. You'll see how to work with new features that make your coding faster and catches errors even before you build your app. Xcode integrates the Git source code repository so that you can easily track your code on your own Mac, a networked Mac, or a public repository such as GitHub.
- ▶ **Hour 5, “Using Compiler Directives”—**Compiler directives help you manage files and use the preprocessor to prepare code for the compiler itself.

## Part II: Working with the Objective-C Basics

This part of the book delves into the concepts of messaging, classes, selectors, building blocks, and memory.

- ▶ **Hour 6, “Exploring Messaging and a Testbed App”**—Messaging is the heart of Objective-C. You send messages rather than call methods or functions, and the difference between Objective-C and other languages is significant.
- ▶ **Hour 7, “Declaring a Class in an Interface File”**—This hour explores the basics of declaring a class. You see how to import other files and use forward declarations for classes and protocols.
- ▶ **Hour 8, “Declaring Instance Variables in an Interface File”**—You can assign instance variables to classes. You also see how to deal with static typing and the scope of instance variables.
- ▶ **Hour 9, “Declaring Properties in an Interface File”**—Objective-C lets you declare properties of a class in your interface file rather than declaring variables. The property declaration provides more features than a simple C variable declaration. Furthermore, the compiler can synthesize the accessors for the property based on the property declaration.
- ▶ **Hour 10, “Declaring Methods in an Interface File”**—Method declarations in Objective-C provide the same functionality that they do in other languages, but the details in this message-based environment are different.
- ▶ **Hour 11, “Declaring Actions in an Interface File”**—Actions are typically triggered by user actions in the interface. You declare them in your interface files and manage their interactions with Interface Builder.
- ▶ **Hour 12, “Routing Messages with Selectors”**—The target-action pattern is the basis for message routing; it is explained in this hour. You also see how to use selectors and the SEL data type.
- ▶ **Hour 13, “Building on the Foundation”**—Throughout Objective-C, you find certain patterns repeated over and over. These include its data types and the concept of mutable and immutable versions of them.
- ▶ **Hour 14, “Defining a Class in an Implementation File”**—After all those declarations, you get down to actually defining the code behind the declarations in this hour.
- ▶ **Hour 15, “Organizing Data with Collections”**—Arrays are one of the core features of programming languages. Objective-C adds additional collection objects, including dictionaries and sets. This hour covers how to use them.



- ▶ **Hour 16, “Managing Memory and Runtime Objects”**—Whether you are running on a mainframe or supercomputer or a relatively small mobile device, memory is a critical resource. There are many tools to help you manage memory, but in most cases, you still need to do something and understand what your choices are. This hour introduces you to Automatic Reference Counting (ARC), the modern way to manage Objective-C memory; it also shows you how garbage collection has been used in the past.

## Part III: Expanding and Extending Classes

One of the differences between Objective-C and other object-oriented languages is that you can expand and extend classes in a variety of ways and not just by subclassing them.

- ▶ **Hour 17, “Extending a Class with Protocols and Delegates”**—Next to the ability to override classes, protocols and delegates are the most commonly used way for adding functionality to a variety of classes.
- ▶ **Hour 18, “Extending a Class with Categories and Extensions”**—Categories and extensions let you add methods and even variables to existing classes—even those for which you might not have the source code.
- ▶ **Hour 19, “Using Associative References and Fast Enumeration”**—These relatively recent additions to Objective-C enable you to become a power programmer. You can add and remove associations at runtime, which gives you more flexibility with your instance variables. Fast enumeration also speeds your runtime performance and might mean that you have less code to write yourself.
- ▶ **Hour 20, “Working with Blocks”**—Blocks provide the ability to create and use sections of code that are portable and anonymous. You can use them as arguments of methods or functions, typically as completion handlers for completion routines, handlers, and errors, as well as for specific types of functionality, such as sorting and view manipulation. The concept is borrowed from languages such as C, Ruby, Python, and Lisp; it has been part of Objective-C since Mac OS X 10.6 (Snow Leopard) and iOS 4.0.

## Part IV: Beyond the Basics

Exception and error handling, queues and threading, the debugger, and Xcode debug gauges allow you to analyze and improve your projects and their performance.

- ▶ **Hour 21, “Handling Exceptions and Errors”**—As in many languages, you can set up try/catch/finally structures to catch exceptions as close as possible to the place where they happen. In this hour, you see how to use them and how to integrate them with the debugger.

- ▶ **Hour 22, “Grand Central Dispatch: Using Queues and Threading”**—Even on small mobile devices, you have the ability to manage multiprocessing. This hour shows you the techniques that are available for multiprocessing.
- ▶ **Hour 23, “Working with the Debugger”**—During development with Xcode, you can use the debugger and console to improve your code. This hour shows you the basics and explores break points, preferences, and some advanced debugging techniques.
- ▶ **Hour 24, “Using Xcode Debug Gauges for Analysis”**—Although debugging can help you get rid of bugs, you need to monitor your app’s performance. Memory leaks aren’t bugs unless they crash your app, but with these tools you can spot them before the damage is done. Debug gauges are a simple graphical interface on top of the more sophisticated Instruments app; both are discussed here.

## Part V: Appendixes

The appendixes provide additional reference material that can help you with the concepts in the book. There is even more material on [developer.apple.com](http://developer.apple.com) and on the author’s website at [northcountryconsulting.com](http://northcountryconsulting.com).

- ▶ **Appendix A, “C Syntax Summary”**—Objective-C is a thin layer built on top of C. This appendix reviews the basic C syntax that matters for Objective-C. Much of it is reimplemented for you in the frameworks of Cocoa, so this appendix walks you through what is left.
- ▶ **Appendix B, “Apps, Packages, and Bundles”**—Your app on Mac OS X or iOS is actually a collection of files, including code, resources such as data stores and images, and property lists. This is a high-level overview so that you can understand what it is you are building with Xcode and Objective-C.
- ▶ **Appendix C, “Archiving and Packaging Apps for Development and Testing”**—This overview gives you the basic information you need to know to share your apps with testers and through the App Store. It is not just a matter of copying files.

*This page intentionally left blank*

# HOUR 3

## Using Object-Oriented Features in Objective-C

---

### What You'll Learn in This Hour

- ▶ Sending and receiving messages
- ▶ Looking inside a message
- ▶ Initializing objects

## Communicating to Methods with Messages

Perhaps the biggest difference between Objective-C and languages such as C++ is its messaging syntax as well as the way people talk about it. Objective-C has classes just as other object-oriented languages do, and those classes can have methods within them. You communicate with those methods with messages. A message is enclosed within square brackets, and it consists of the name of the object to which it is being sent followed by the message itself.

The implementation files that you create carry the `.m` suffix because originally they were referred to as message files that contained the code for the messages defined in the header (`.h`) files. (This is possibly an apocryphal tale, but the importance of messaging in Objective-C is undisputed.)

### NOTE

---

There is an ulterior purpose to this section. It does help you to understand how to communicate with Objective-C methods and messages, but the examples begin with the messages that you use to allocate and initialize objects.

---

## Looking at a Simple Message

Here is a simple message that is sent to an object called `myObject`, which is assumed to be of type `NSObject`—the object that is the root class of most class hierarchies in Objective-C.

```
[myObject init];
```

This message calls the `init` method on `myObject`.

## NOTE

---

That is the point made previously about the different way many people talk about Objective-C methods: They often refer to calling a message “on” an object.

---

Methods can return a value. If a method returns a value, you can set it to a variable as in the following:

```
myVariable = [myObject init];
```

## Declaring a Method

When you declare a simple method, you use an Objective-C variation on C function syntax. `NSObject`, the root class of almost all of the Objective-C objects you use, does declare an `init` method.

## NOTE

---

### `NSObject` Class

As a matter of simplification, `NSObject` is referred to as the superclass of all objects in the Objective-C frameworks for Cocoa and Cocoa Touch. Note the `NS` prefix, which indicates that this method dates back to NeXTSTEP. Although it does not matter in most cases, the fact that `NSObject` is only the superclass of objects in the frameworks (and not all Objective-C objects) is something to bear in mind.

---

The following is the declaration that supports the messages shown in the previous section:

```
- (id) init
```

As you might surmise, the `init` method shown here returns a result of type `id`. (You find out more about `id` shortly.)

The minus sign at the start of the method is an important part of the declaration: It is the method type. It indicates that this is a method that is defined for instances of a class. Any instance of the class in which this declaration is used can invoke this method.

To put it another way, you (or an instance of a class) can send the `init` message to any instance of this class. Because this is the `NSObject` superclass of every other object, that means you can send the `init` message to any instance of any class.

There is more on allocating and initializing objects later in this hour.

## Using Class Methods

The minus sign at the start of the method shown in the previous section indicates that it is an instance method. There is another type of method in Objective-C: a class method. It is indicated by a plus sign.

### TIP

---

Class objects are often used as *factory objects* to create new, concrete instances of themselves.

---

A message to an instance method can be sent to any instance of that class subject to constraints for that specific class. Whereas you call an instance method on an instance of a class, you call a class method on the class itself. No instance is involved.

Class methods are used most frequently as *factory methods*. Perhaps the most common class method is `alloc`. For `NSObject`, its declaration is

```
+ (id)alloc;
```

Whereas you send `init` to an instance, as in this case:

```
[myObject init];
```

`alloc` allocates an uninitialized instance of a class as in

```
[MyClass alloc];
```

This returns an instance of `MyClass`. As you can see in the declaration, this result is of type `id`. It is time to explore that type.

## Working with `id`—Strongly and Weakly Typed Variables

Objective-C supports strongly and weakly typed variables. When you reference a variable using a strong type, you specify the type of the variable. The actual variable must be of that type or a subclass of that type; if it is a subclass, it is, by definition, the type of all of its superclasses.

In Cocoa, you can declare a variable as:

```
NSArray *myArray
```

This means you could be referring to an object of type `NSMutableArray`, which is a subclass. You can write the same code to work with elements of the array no matter what its actual type is. If necessary, you might have to coerce a specific instance to the subclass that you want (if you know that is what it is).

`id` is the ultimate weakly typed variable; it could be any class. That is why it is used as the return type from `alloc`. `alloc` is a class method on `NSObject` so if you call it on an `NSArray`, you get an instance of `NSArray` returned through `id`.

## NOTE

---

### `instancetype`

A new keyword, `instancetype`, is available when working with related result types. By convention, methods beginning with names such as `init` and `alloc` always return objects that are an instance of the class type that receives the message. Thus, `[MyClass init]` returns an instance of `MyClass`. `init` is declared as returning an `id`, but with this convention it is the related type. You can now use `instancetype` instead of `id` to specify that the returned value is the same type as the receiver's type (that is, `MyClass` in this example). Because `instancetype` turns out to be more specific than `id` at compile time, it is preferable to use in these cases because it can allow for more error checking. As you can see from the examples in this hour, many types of the code you write already use specific variables of the correct type; however, `instancetype` in declarations of your own methods can make that more likely.

---

## Nesting Messages

Messages can be nested within one another. You could write the following:

```
myObject = [MyClass alloc];  
myObject = [myObject init];
```

This would use the class method of `MyClass` to allocate a new instance of `MyClass`, which you immediately assign to `myObject`.

You can nest them together as follows:

```
myObject = [[MyClass alloc] init];
```

The rules for nesting square brackets are the same as for nesting parentheses—the innermost set is evaluated first.

## TIP

---

Xcode gives you a big assist by enabling you to set a preference to match brackets as you type.

---

## Looking at Method Signatures and Parameters

`alloc` and `init` are very good basic examples because they have no parameters. Most methods in any language do have parameters. For example, you can write an area function that takes two parameters (height and width) and returns the product as its return value.

Other languages generally specify a name and a type for each parameter, and so does Objective-C. However, it adds another dimension: It labels each parameter.

This labeling means that the code is more readable, but you do have to understand what is going on when there is more than one parameter. When there is no parameter, the message is simply the receiver and the name of the method:

```
[myObject init];
```

If there is a parameter, it follows the method name. In the message, a colon precedes the parameter itself. For example, in `NSSet`, you can initialize a set with an `NSArray` using code like this:

```
mySet = [NSSet alloc];  
[mySet initWithArray: myArray];
```

The declaration needs to specify not only the parameter name, which is used in the code of the method, but also its type:

```
(instancetype) initWithArray: (NSArray *) array;
```

The second and subsequent parameters are also labeled. The difference is that the first parameter is labeled in effect by the name of the method. If you add more parameters, their names and types are needed; they are preceded by a keyword (which, in the case of the first parameter is the method name). Here is another `NSSet` method. It initializes an `NSSet` to the elements of another `NSSet` (the first parameter). The second parameter specifies whether the elements of the first set are to be copied or not.

Here is a typical invocation:

```
[mySet: initWithSet: aSet copyItems:YES];
```

Here is the declaration:

```
(instancetype) initWithSet: (NSSet *) set copyItems: (BOOL) flag
```

In documentation (and in this book), the signature sometimes is compressed to be the result, method name, and parameters so that the previous declaration is shown as

```
(instancetype) initWithSet:copyItems:
```

## Allocating and Initializing Objects

The messages shown in this hour have demonstrated how you can allocate and initialize objects. Because these processes are used so extensively in Objective-C, it is worthwhile to take a few moments to look more carefully at these messages.



As noted previously, you most often create instances of classes by calling `alloc`. This is a class method of `NSObject`, which means it is available to any class in the frameworks or in your own code. You simply send a message to the class, and you get back a new instance.

```
MyClass *newInstance = [MyClass alloc];
```

Immediately thereafter, you initialize the newly created instance, most often with a call to `init` or a related method.

```
newInstance = [newInstance init];
```

As noted previously, you can combine these two messages into one line of code.

```
MyClass *newInstance = [[MyClass alloc] init];
```

It makes sense that `alloc` returns an object that you can place in an instance variable. You might be surprised that `init` also returns an object. The reason for this is that after you have created the new object, the call to `init` might not only set instance variables and other settings, but it also might decide to replace the allocated object with another one. That is the one that you should use thereafter.

By convention, initializer names start with `init`. They might have additional parameters, such as these various initializers for `UIBarButtonItem` in Cocoa Touch:

```
- initWithBarButtonSystemItem:target:action:
- initWithCustomView:
- initWithImage:style:target:action:
- initWithTitle:style:target:action:
- initWithImage:landscapeImagePhone:style:target:action:
```

Initializers may set various properties of the object being created; they also may take various routes to complete the initialization. If you create a series of initializers, there are two rules to follow:

- ▶ The first step in an initializer is to call `[super init]`. This ensures that the basic object is there and complete.
- ▶ Initializers can call other initializers that each performs some initialization. The most complete initializer (that is, the one that sets all of the properties rather than just some of them) is the designated initializer. The others are secondary initializers.

It is a good practice to put all of the initializers together in your code, possibly using a `#pragma mark -` section. This separates the initializers and, in the list of methods in the jump bar for a file, the pragma section names will appear.

## Summary

In this hour, you have seen the messaging structure that is at the heart of Objective-C. It is not just another way of talking about the calling of methods that other languages use; it is a different way of constructing software. You see in later hours how the messaging architecture helps to implement dynamic aspects of Objective-C programs.

The common `alloc` and `init` methods have been used in this hour as examples. You have also seen how you can work with them and construct a hierarchy of designated and secondary initializers to construct your runtime objects.

## Q&A

**Q.** Why does `init` return an `id`?

**A.** By returning an `id`, which is often a reference to the object on which it was called, it allows `init` to substitute a new object for the original one. Increasingly, the declarations are being changed to use the new `instancetype` return value, which allows for more error checking. Each new release of iOS adds more of these revisions.

**Q.** What is the difference between a class method and an instance method?

**A.** You must have an instance of a class to use an instance method; you can simply call a class method with the class name (that is, you do not need an instance). Class methods are often used as factory methods to create new instances of the class.

## Workshop

### Quiz

1. How do you differentiate class and instance methods in your code?
2. When do you call `[super init]` in an initializer?

### Quiz Answers

1. Class methods begin with `+`; instance methods begin with `-`.
2. This is the first thing you do. You might go up through a hierarchy, but you have to have all of the superclasses created and initialized before you continue with your own.

### Activities

Browse the Cocoa frameworks to see how messages are structured. Get used to the sequence of labeled parameters in the messages. It might take a while to be comfortable reading them, but remember that this is the environment in which you are going to be working.

*This page intentionally left blank*

# Index

## Symbols

- @ (at symbol), 74
- ^ (carat), 264
- : (colon), 145
- (hyphen), 142
- + (plus sign), 35, 142
- [] (square brackets), 94

## A

- abstract classes, 106
- abstraction, 24-25
- accessing
  - objects in collections, 218
  - properties
    - with dot syntax, 134-135
    - with message syntax, 133
- accessors
  - creating with @synthesize, 138
  - explained, 28, 136, 226

## actions

- building apps with, 150-159
  - iOS apps, 155-157
  - OS X apps, 151-154
- disconnecting, 161
- headers, 159
- IBAction, 159
- OS X versus iOS, 159-161
- overview, 107, 149-150
- removing, 161
- setting breakpoints on, 160
- addObserver method, 186
- addObserverForName method, 186
- allObjects method, 215
- alloc method, 35, 37-38, 113
- allocating objects, 37-38
- anonymous categories. *See* extensions
- anyObject method, 218
- AppDelegate class
  - .h files, 70-71
  - .m files, 71-73

**Applcon.appiconset folder, 318**

**AppKit, 179**

**Apple developer program**

- Apple IDs, registering for, 10
- choosing, 8-10
- enrolling in, 8
- iOS program, 9-10
- Mac OS program, 9-10
- membership categories, 9-10
- Safari program, 9

**Apple IDs, registering for, 10**

**Apple LLVM Compiler 3.0, 223**

**AppleScript, 241**

**Application and Foundation Kits, 7**

**applicationDidFinishLaunching  
WithOptions method, 201**

**Applications folder, 41**

**apps**

- archiving, 321-324
  - iOS archives, 324
  - Mac archives, 322-323
- building with actions, 150-159
  - iOS apps, 155-157
  - OS X apps, 151-154
- bundles
  - asset catalogs, 318
  - definition of, 317
  - explained, 317-318
  - lproj files, 318
  - plist files, 319
  - precompiled header files  
(.pch), 319
  - reasons for using, 318
- Instruments, 310

testbed apps. *See also*  
testbed apps

- building on iOS, 78
- building on OS X, 79
- creating, 193-196
- message syntax, 94-95
- overview, 77-78
- text fields, 78-94

**ARC (automatic reference  
counting)**

- declared property attributes,  
226-227
- overview, 113, 225-227

**archiving apps, 321-324**

- iOS archives, 324
- Mac archives, 322-323

**arguments**

- overview, 145
- setting, 174

**arrays, 315**

- accessing objects in, 218
- creating, 215-217
- enumerating, 215-217
- iterating through, 206
- reading and writing, 213
- testing membership in, 217

**arrayWithArray method, 210**

**arrayWithContentsOfFile  
method, 213**

**arrayWithContentsOfURL  
method, 213**

**arrayWithObject method, 210**

**arrayWithObjects method, 210**

**asset catalogs, 318**

**associative references, 250-251**

- adding, 251
- getting and setting, 252

hiding in declared property,  
252

removing, 254

**asynchronous tasks, adding to  
queues, 289**

**atomicity, 137, 226**

**attributes of declared properties**

- accessor methods, 136
- atomicity, 137
- other attribute decorators,  
137
- overview, 133, 135
- setter semantics, 137
- writability, 136

**automatic reference counting  
(ARC), 113, 225-227**

**@autoreleasepool directive, 228**

**\_\_autoreleasing— variable  
qualifier, 227**

**autoreleasing variables, 228**

**Availability.h file, 319**

## B

**BankAccount class, 26-28**

**base objects, overriding, 29**

**Base.lproj file, 318**

**battery usage, 310**

**binding (dynamic), 117**

**\_\_block storage type modifier,  
269**

**block variables, creating, 264-265**

**Block\_copy() function, 269**

**Block\_release() function, 269**

**blocks**

- compared to callback functions, 264
  - in Cocoa, 266
- copying to heap, 269
- creating as block variables, 264-265
- creating from callback functions, 265
- enumeration methods, 265-267
  - enumerateKeysAndObjectsUsingBlock:, 268
  - enumerateLinesUsingBlock:, 267-268
  - enumerateObjectsUsingBlock:, 268
- overview, 259-260
- try/catch blocks, 275-278

**branches**

- creating, 50-56
- overview, 47-48

**breakpoints, 293, 297**

- breaking on conditions, 298-302
- editing, 298-300
- setting on actions, 160

**building apps with actions, 150-159**

- iOS apps, 155-157
- OS X apps, 151-154

**bundles. See also packages**

- asset catalogs, 318
- definition of, 317
- explained, 317-318
- lproj files, 318
- plist files, 319

- precompiled header files (.pch), 319
- reasons for using, 318

**Burroughs large systems, 132****business membership (developer program), 10****C****C language**

- control structures, 315
  - if statements, 315
  - repeat statements, 316
  - switch statements, 315
- data types, 313
  - arrays, 315
  - enumerated type, 313
  - pointers, 314
  - struct, 314
- development of, 24

**The C Programming Language (2nd Edition) (Kernighan and Ritchie), 24, 313****C++, 24****calculate method, 133****callback functions, 260-261**

- compared to blocks, 264
- creating blocks from, 265

**calling methods, 104****callStackSymbols, 296****carat (^), 264****catalogs (asset), 318****@catch, 275-278****catching exceptions, 280****categories, 30**

- alternatives to, 240
- compared to protocols, 239
- compared to subclasses, 241-242
- confining modifications to, 242
- creating, 242-245
- informal protocols, 246
- when to use, 240

**changes**

- discarding, 56
- tracking, 52-55

**checking out repositories, 60****choosing developer program, 8-10****class clusters, 183-184****@class directive, 74, 108, 117****class hierarchy, modifying, 241-242****class name, creating instance variables with, 117-119****classes, 25. See also protocols**

- abstract classes, 106
- AppDelegate
  - .h files, 70-71
  - .m files, 71-73
- associative references, 250-251
  - adding, 251
  - getting and setting, 252
  - hiding in declared property, 253
  - removing, 254
- BankAccount, 26-28

- categories, 240
  - alternatives to, 240
  - anonymous categories, 242-246
  - compared to protocols, 239
  - compared to subclasses, 241-242
  - confining modifications to, 242
  - creating, 242-245
  - informal protocols, 246
  - when to use, 240
- class hierarchy, modifying, 241-242
- collection classes, 112-114, 209-210. *See also* collections
- creating
  - preparation, 98-99
  - with Xcode, 99-103
- CurrencyConverter
  - convertCurrency: method, 191-193
  - convertUnits: method, 193
  - convertUnits:withFactor: method, 194-200
  - creating instance variables with class name, 117-119
  - creating instance variables with superclass name, 119-123
  - creating with Xcode, 99-103
  - CurrencyConverter.h file, 99-103
  - CurrencyConverter.m file, 104
  - designing, 190-191
  - history of, 98
  - init methods, 200-202
  - instance variables, creating with id, 114-117
  - declaring
    - basic class declaration, 106-107
    - forward references, 107-108
  - designing, 97-98
  - extensions, 245-246
  - Foundation framework
    - class clusters, 183-184
    - class groups, 180-182
    - mutability, 182-183
    - root classes, 180
    - turning into protocols, 181
  - instantiating, 111-112
  - ARC (automatic reference counting), 113
  - choosing context, 112-114
  - placing class instances in context, 112
  - methods
    - actions, 107
    - addObserver, 186
    - addObserverForName, 186
    - allObjects, 215
    - alloc, 35, 37-38, 113
    - anyObject, 218
    - applicationDidFinishLaunchingWithOptions, 201
  - arrayWithArray, 210
  - arrayWithContentsOfFile, 213
  - arrayWithContentsOfURL, 213
  - arrayWithObject, 210
  - calculate, 133
  - class methods, 35
  - class methods versus instance methods, 142-143
  - containsObject, 215
  - convertCurrency, 191-193
  - convertUnits, 193, 201
  - convertUnits:withFactor:, 194-200
  - dealloc, 113, 223
  - declaring, 34, 143-146
  - description, 104
  - dictionaryWithContentsOfFile, 214
  - dictionaryWithContentsOfURL, 214
  - dictionaryWithDictionary, 210
  - dictionaryWithObject, 212
  - dictionaryWithObjects, 212
  - didFinishLaunchingWithOptions, 244, 262
  - draw, 104
  - enumerateKeysAndObjectsUsingBlock, 260, 268
  - enumerateLinesUsingBlock, 260, 267-268
  - enumerateObjectsUsingBlock, 260, 268
  - exceptionWithName, 278

- getReturnValue, 174-175
- hiding, 29
- implementing, 194-200
- init methods, 38, 200-202
- initWithBool, 142
- initWithFactor, 201
- instancesRespondToSelector, 176
- invocationWithMethodSignature, 172
- invoke, 175
- invokeWithTarget, 175
- keyEnumerator, 215
- nextObject, 214
- numberWithBool, 142
- objectAtIndex, 218
- objectEnumerator, 215
- objectForKey, 215, 218
- optional methods, 236
- parameters, 36-37
- postNotification, 185
- postNotificationName, 185
- removeObserver, 186
- respondToSelector, 176
- return values, 174-175
- returning multiple result values from, 112-114
- reverseObjectEnumerator, 215
- setArgument, 174
- setByAddingObject, 214
- setByAddingObjectsFromArray, 214
- setByAddingObjectsFromSet, 214
- setSelector, 174
- setTarget, 173
- setter semantics, 137
- setWithArray, 214
- setWithObjects, 215
- setWithSet, 210
- signatures, 36-37, 172
- superclass methods, 104
- syntax, 141
- useTypeDefFunctionPointer, 262
- writeToFile, 213
- writeToURL, 213-214
- naming, 98-99
- NSArray, 209-210
  - creating and enumerating, 215-217
  - enumerateObjectsUsingBlock: method, 268
  - iterating through, 206
- NSAutoreleasePool, 228
- NSDate, 208
- NSDictionary
  - enumerateKeysAndObjectsUsingBlock: method, 268
  - overview, 209-210
- NSEnumerationConcurrent, 266
- NSEnumerationReverse, 267
- NSEnumerator
  - methods, 214-215
  - NSEnumerator instances, creating, 215
  - overview, 256
- NSError, 274-275
- NSException, 272-274
- NSNotification, 183
- NSNotificationCenter, 185
- NSNumber
  - getters, 192
  - overview, 183, 208
- NSObject, 34, 179-180
- NSOperationQueue, 287
- NSProxy, 180
- NSSet, 209-210, 268
- NSString, 209, 267-268
- subclasses
  - compared to categories, 241-242
  - pros and cons, 231-232
  - superclass methods, 104
- cloning repositories, 60**
- clusters (class), 183-184**
- Cocoa frameworks, 179**
  - errors, 277
  - views, 78
  - windows, 78
- Cocoa Touch frameworks, 179**
  - errors, 277
- code property (NSError), 274**
- collections**
  - accessing objects in, 218
  - advantages of, 205-207
  - arrays
    - creating, 215-217
    - enumerating, 215-217
    - reading and writing, 213
  - building from property lists at runtime, 209
  - collection classes, 112-114, 209-210
  - creating with literals, 212
  - creation methods, 210-212
  - dictionaries, reading and writing, 213-214



enumeration  
 NSArray instances, 215  
 NSArray methods, 214-215  
 mutable collections, 207  
 optimization, 211  
 overview, 205-207  
 in property lists, 207-209  
 sets, creating, 214  
 testing membership in, 217

**colon (:), 145**

**color, setting for messages, 92-94**

**compiler directives**

@autoreleasepool, 228  
 @catch, 275-278  
 @class, 74, 108, 117  
 #define, 66-67  
 definition of, 66  
 @dynamic, 139  
 @end, 74, 107  
 @finally, 275  
 #ifdef, 67  
 #ifndef, 67  
 @implementation, 74  
 #import, 68-69  
 #include, 68  
 @interface, 74  
 @optional, 236, 246  
 @protocol, 74, 108  
 @required, 236  
 @selector, 169  
 @synthesize, 130-132, 138  
 @try, 275-278

**concurrency**

definition of, 283  
 dispatch queues, 288  
 adding asynchronous tasks to, 289  
 designing with, 290  
 global concurrent dispatch queues, 288-289  
 GCD (Grand Central Dispatch), 285-286  
 implementing, 284  
 multicore processors, 283-284  
 queues  
 dispatch queues, 287  
 dispatch sources, 287  
 operation queues, 287-288  
 overview, 286  
 threads, 284-285

**conditions, breaking on, 298-302**

**confining modifications to categories, 242**

**console logs**

NSLog, 295-297  
 overview, 293-294  
 troubleshooting with, 294  
 viewing, 294

**containsObject: method, 215**

**control structures, 315**

if statements, 315  
 repeat statements, 316  
 switch statements, 315

**Convert to Objective-C ARC command, 225**

**convertCurrency: method, 191-193**

**convertUnits: method, 193, 201**

**convertUnits:withFactor: method, 194-200**

**copying**

blocks to heap, 269  
 method declarations, 143

**Core Foundation framework, 180**

**counting references. See reference counting**

**Cox, Brad, 7**

**Cox, Brian J., 165, 179**

**CPU utilization, monitoring, 306-307**

**creating. See also declaring**

actions  
 iOS projects, 155-157  
 OS X projects, 151-154  
 arrays, 215-217

associative references, 251

blocks

as block variables, 264-265  
 from callback functions, 265

categories, 242-245

classes

preparation, 98-99  
 with Xcode, 99-103

collections

with creation methods, 210-212  
 with literals, 212

iOS archives, 324

Mac archives, 322-323

- method signatures, 172
  - NSEnumerator instances, 215
  - NSInvocation object, 172
  - sets, 214
  - testbed apps, 193-196
- CurrencyConverter**
- convertCurrency: method, 191-193
  - convertUnits: method, 193
  - convertUnits:withFactor: method, 194-200
  - creating with Xcode, 99-103
  - CurrencyConverter.h file, 99
  - CurrencyConverter.m file, 104
  - designing, 190-191
  - history of, 98
  - init methods, 200-202
  - instance variables
    - creating with class name, 117-119
    - creating with id, 114-117
    - creating with superclass name, 119-123
- cut back, 132**
- D**
- dangling pointers, 227**
- data abstraction, 24-25**
- data types, 313**
- arrays
    - iterating through, 206
    - overview, 315
  - dynamic typing, 190
  - enumerated type, 313
  - id, 114-117, 129
  - pointers, 314
  - primitive types, 205
  - SEL, 168-169
  - struct, 314
- dealloc method, 113, 223**
- debug gauges, 305-306**
- monitoring CPU utilization, 306-307
  - monitoring energy, 308-309
  - monitoring memory utilization, 307-308
- debugger**
- breakpoints, 293, 297
    - breaking on conditions, 298-302
    - editing, 298-300
  - console logs
    - NSLog, 295-297
    - overview, 293-294
    - troubleshooting with, 294
    - viewing, 294
  - documentation, 302
- debugging, 302**
- debug gauges, 305-306
    - monitoring CPU utilization, 306-307
    - monitoring energy, 308-309
    - monitoring memory utilization, 307-308
  - debugger
    - breakpoints, 293, 297-302
    - console logs, 293-297
    - documentation, 302
- with description method, 104
  - with Instruments app, 310
- declared properties**
- adding, 201
  - attributes, 226-227, 253
- declaring. See also creating**
- categories, 242-245
  - classes
    - basic class declaration, 106-107
    - forward references, 107-108
  - instance variables, 107, 111
  - methods, 34, 143
    - arguments, 145
    - method names, 144-145
    - return values, 143-144, 146-147
    - writing method declarations, 146
  - properties
    - attributes, 133
    - example, 130-133
    - memory management, 132
  - variables
    - basic variable declaration, 128-130
    - legacy instance variable declarations, 138
- #define directive, 66-67**
- delegates, 264**
- Master-Detail Application template example, 232-235
  - overview, 233
- description method, 104**
- design patterns, 149**

**designing**

- classes, 97-98
- CurrencyConverter, 190-191
- with queues, 290

**Developer folder, 41****developer program**

- Apple IDs, registering for, 10
- choosing, 8-10
- enrolling in, 8
- iOS program, 9-10
- Mac OS program, 9-10
- membership categories, 9-10
- Safari program, 9

**Developer Technical Support (DTS), 9****development of Git, 48****development of Objective-C, 7-8****dictionaries**

- accessing objects in, 218
- reading and writing, 213-214
- testing membership in, 217

**dictionaryWithContentsOfFile method, 214****dictionaryWithContentsOfURL method, 214****dictionaryWithDictionary method, 210****dictionaryWithObject method, 212****dictionaryWithObjects method, 212****didFinishLaunchingWithOptions method, 244, 262****directives**

- @autoreleasepool, 228
- @catch, 275-278
- @class, 74, 108, 117
- #define, 66-67

definition of, 66

@dynamic, 139

@end, 74, 107

@finally, 275

#ifdef, 67

#ifndef, 67

@implementation, 74

#import, 68-69

#include, 68

@interface, 74

@optional, 236, 246

@protocol, 74, 108

@required, 236

@selector, 169

@synthesize, 130-132, 138

@try, 275-278

**discarding changes with Git, 56****disconnecting actions, 161****dispatch queues, 288**

- adding asynchronous tasks to, 289

designing with, 290

explained, 287

global concurrent dispatch queues, 288-289

**dispatch sources, 287****distributed objects, 182****documentation for debugger, 302****domain codes (NSError), 274****domain property (NSError), 274****dot syntax, accessing properties with, 134-135****downloading Xcode, 41****draw method, 104****DTS (Developer Technical Support), 9****dynamic binding, 117****@dynamic directive, 139****dynamic typing, 190****dynamism, 273****E****Edit Breakpoint menu, 299****editing breakpoints, 298-300****encapsulation**

- overview, 25-28
- variable declarations, 129-130

**@end directive, 74, 107****energy, monitoring, 308-309****Energy Impact gauge, 308-309****enterprise program (developer program), 10****enumerated type, 313****enumerateKeysAndObjects UsingBlock method, 260, 268****enumerateLinesUsingBlock method, 260, 267-268****enumerateObjectsUsingBlock method, 260, 268****enumeration,**

- arrays, 215-217
- block enumeration methods, 265-267
  - enumerateKeysAndObjects UsingBlock, 268
  - enumerateLinesUsingBlock, 267-268
  - enumerateObjectsUsingBlock, 268

- collections
  - NSEnumerator instances, 215
  - NSEnumerator methods, 214-215
- fast enumeration, 254-256
  - with NSEnumerator, 256
  - without NSEnumerator, 255
- NSEnumerationConcurrent, 266
- NSEnumerationReverse, 267
- small objects, 267
- error codes (NSError), 276**
- errors**
  - in Cocoa and Cocoa Touch, 276-277
  - compared to exceptions, 272
  - NSError class, 274-275
  - overview, 271-272
- evolution of Objective-C, 7-8**
- exceptions**
  - catching, 280
  - compared to errors, 272
  - identifying, 277-278
  - NSException class, 272-274
  - overview, 271-272
  - throwing, 278-279
- exceptionWithName method, 278**
- expressions, logging, 296**
- extensions, 30, 245-246**

**F**

- fast enumeration, 254-256**
  - with NSEnumerator, 256
  - without NSEnumerator, 255

- fields, text, 81**
  - adding in iOS, 82-84
  - adding in OS X, 85-87
  - connecting to code in iOS, 88-91
  - connecting to code in OS X, 91
  - sending messages to, 92-94
- \_\_FILE\_\_ macro, 295**
- file system classes, 181**
- files**
  - Availability.h, 319
  - creating dictionaries from, 214
  - .h files, 33, 66, 70-71
  - implementation files. *See* implementation files, defining classes in
  - lproj, 318
  - .m files, 33, 66, 71-73
  - main.m file, 69-70
  - missing files, 65
  - .pch files, 66
  - plist, 319
  - plist files, 66
  - precompiled header files (.pch), 319
  - prefix headers, 67-69
  - .rtf files, 65
  - .strings files, 65
  - writing arrays to, 213
  - .xib files, 65
- @finally, 275**
- folders**
  - AppIcon.appiconset, 318
  - Applications, 41
  - Developer, 41

- Images.xcassets, 318
- UIImage.launchImage, 318
- forks, 47-48**
- for loops, adding to breakpoints, 298-302**
- forward references, 107-108**
- Foundation framework**
  - classes
    - class clusters, 183-184
    - class groups, 180-182
    - root classes, 180
    - turning into protocols, 181
  - Core Foundation framework, 180
  - layers, 23-24
  - mutability, 182-183
  - notifications, 184-186
    - posting, 185
    - registering to receive, 186
    - removing observers, 186
    - values, 183-185
  - overview, 22, 179-180
- \_\_func\_\_ macro, 295**
- functions. *See also* methods**
  - Block\_copy(), 269
  - Block\_release(), 269
  - callback functions, 260-261
    - compared to blocks, 264
    - creating blocks from, 265
  - objc\_getAssociatedObject, 251-252
  - objc\_removeAssociatedObjects, 251
  - objc\_setAssociatedObject, 250, 251, 254
  - typedef function pointer, 261-262

**G**

garbage collection, 222

GCD (Grand Central Dispatch), 285-286

getReturnValue method, 174-175

Git, 11, 50-56

- discarding changes, 56
- history of, 48
- remote repositories, 58-60
- tracking changes, 51-55

global concurrent dispatch queues, 288-289

Grand Central Dispatch (GCD), 285-286

groups of classes, 180-182

**H**

.h files, 33, 66, 70-71

headers

- action headers, 159
- .h files, 70-71
- precompiled header files (.pch), 319
- prefix headers, 67-69

heap, copying blocks to, 269

hiding

- associative references in declared property, 253
- methods, 29

history of Git, 48

history of Objective-C, 7-8, 249-250

hyphen (), 142

**I**

IBAction, 159

IBOutlet, 137

id data type, 114-117, 129

id variable, 35

identifying exceptions, 275-278

if statements, 315

#ifdef directive, 67

#ifndef directive, 67

imageset folders, 318

Images.xcassets folder, 318

@implementation directive, 74

implementation files, defining classes in

- CurrencyConverter, 190-191
  - convertCurrency: method, 191-193
  - convertUnits: method, 193
- init methods, 200-202
- method implementation, 197-200
- Testbed app, 193-196

implementing

- concurrency, 284
- methods, 194-200
- object-oriented programming, 21-22
- properties, 138-139
- selectors, 170-171

#import directive, 68-69

#include directive, 68

informal protocols, 246

inheritance, 29-30

init methods, 38, 200-202

initializers, 37-38

initializing objects, 37-38

initWithBool method, 142

initWithFactor method, 201

inline function pointers, 261

instance methods, versus class methods, 142-143

instance variables. *See also* associative references

- compared to properties, 127
- creating for CurrencyConverter
  - with class name, 117-119
  - with id, 114-117
  - with superclass name, 119-123
- declaring
  - explained, 107, 111
  - legacy instance variable declarations, 138
- visibility, 123

instances

- overview, 25
- testing whether instance can respond to selector, 175-176

instancesRespondToSelector method, 176

instanceType keyword, 36

instantiating classes, 111-112

- ARC (automatic reference counting), 113
- choosing context, 112-114
- placing class instances in context, 112

Instruments, 310

Interface Builder, selectors with, 171

@interface directive, 74

interprocess communication (IPC), 182

invokeWithMethodSignature method, 172

invoke method, 175

invokeWithTarget method, 175

invoking NSInvocation object, 175

iOS developer program, 9-10

iOS projects

- actions
  - building projects with, 155-157
  - OS X versus iOS, 159-161
- iOS archives, creating, 324
- project files, 65-66
- structure of, 64
- testbed apps
  - adding text fields, 82-84
  - building, 78
  - connecting text fields to code, 89-91
  - creating, 193-196
  - message syntax, 94-95
  - sending messages to text fields, 92-94

IPC (interprocess communication), 182

iterating through arrays, 206

ivars. *See* instance variables

## J-K

Jobs, Steve, 7

Kay, Alan, 165

Kernighan, Brian W., 24

keyEnumerator method, 215

keywords

- instanceType, 36
- strong, 226
- unsafe\_unretained, 227
- weak, 226

## L

language services, 182

LaunchImage.launchimage folder, 318

layers (frameworks), 23-24

legacy instance variable declarations, 138

\_\_LINE\_\_ macro, 295

literals, creating collections with, 212

localization, lproj files, 318

locks, synchronizing with, 287

logs

- NSLog, 295-297
- overview, 293-294
- troubleshooting with, 294
- viewing, 294

loops, for loops, 298-302

Love, Tom, 7, 165, 179

lproj files, 318

## M

.m files, 33, 66, 71-73

Mac archives, creating, 322-323

Mac OS developer program, 9-10

macro definition directive, 66-67

macros, logging, 295

main.m file, 69-70

managing memory. *See* memory management

manual reference counting, 222-225

membership categories (developer program), 9-10

membership in collections, testing, 217

memory leaks, 306, 310

memory management

- autoreleasing variables, 228
- blocks, copying to heap, 269
- garbage collection, 222
- for objects, 132
- overview, 221-222
- reference counting
  - ARC (automatic reference counting), 225-227
  - explained, 222
  - manual reference counting, 222-225
  - variable qualifiers, 227

memory utilization, monitoring, 307-308

messages

- accessing properties with, 133
- adding to breakpoints, 298-300
- color, setting, 92-94
- message syntax, 94-95
- nesting, 36
- overview, 33, 165-166
- receivers, 166

- release, 224
- retain, 224, 228
- selectors
  - creating from string, 169-170
  - creating with @selector, 169
  - implementing, 170-171
  - with Interface Builder, 171
  - overview, 166
  - performSelector, 169-171
  - SEL data type, 168-169
  - setting, 174
  - testing whether instance can respond to selector, 175-176
  - sending to text fields, 92-94
  - signatures, creating, 172
  - simple example, 33-34
  - structure of, 92
- methods. See also functions;**
- protocols**
  - accessors
    - creating with @synthesize, 138
    - overview, 28, 136, 226
  - actions, 107
  - addObserver, 186
  - addObserverForName, 186
  - allObjects, 215
  - alloc, 35, 37-38, 113
  - anyObject, 218
  - applicationDidFinishLaunchingWithOptions, 201
  - arrayWithArray, 210
  - arrayWithContentsOfFile, 213
  - arrayWithContentsOfURL, 213
  - arrayWithObject, 210
  - calculate, 133
  - class methods, 35
  - class methods versus instance methods, 142-143
  - containsObject, 215
  - convertCurrency, 191-193
  - convertUnits, 193, 201
  - convertUnits:withFactor:, 194-200
  - dealloc, 113, 223
  - declaring, 34, 143
    - arguments, 145
    - method names, 144-145
    - return values, 143-144, 146-147
    - writing method declarations, 146
  - description, 104
  - dictionaryWithContentsOfFile, 214
  - dictionaryWithContentsOfURL, 214
  - dictionaryWithDictionary, 210
  - dictionaryWithObject, 212
  - dictionaryWithObjects, 212
  - didFinishLaunchingWithOptions, 244, 262
  - draw, 104
  - enumerateKeysAndObjectsUsingBlock, 260, 268
  - enumerateLinesUsingBlock, 260, 267-268
  - enumerateObjectsUsingBlock, 260, 268
  - exceptionWithName, 278
  - getReturnValue, 174-175
  - hiding, 29
  - implementing, 194-200
  - init methods, 38, 200-202
  - initWithBool, 142
  - initWithFactor, 201
  - instancesRespondToSelector, 176
  - invocationWithMethodSignature, 172
  - invoke, 175
  - invokeWithTarget, 175
  - keyEnumerator, 215
  - nextObject, 214
  - numberWithBool, 142
  - objectAtIndex, 218
  - objectEnumerator, 215
  - objectForKey, 215, 218
  - optional methods, 236
  - parameters, 36-37
  - postNotification, 185
  - postNotificationName, 185
  - removeObserver, 186
  - respondToSelector, 176
  - return values, getting, 174-175
  - returning multiple result values from, 112-114
  - reverseObjectEnumerator, 215
  - setArgument, 174
  - setByAddingObject, 214
  - setByAddingObjectsFromArray, 214
  - setByAddingObjectsFromSet, 214
  - setSelector, 174
  - setTarget, 173
  - setter semantics, 137

- setWithArray, 214
- setWithObjects, 215
- setWithSet, 210
- signatures, 36-37, 172
- superclass methods,
  - calling, 104
- syntax, 142
- useTypeDefFunctionPointer,
  - 262
- writeToFile, 213
- writeToURL, 213-214

#### missing files, 65

modifying class hierarchy, 241-242

#### monitoring

- CPU utilization, 306-307
- energy, 308-309
- with Instruments app, 310
- memory utilization, 307-308

#### multicore processors, 283-284

multiple result values, returning from methods, 112-114

#### mutability, 182-183

#### mutable classes, 182-183

#### mutable collections, 207

#### MyApp-Info.plist file, 319

## N

name property (NSException), 274

#### Name value (NSNotification), 183

#### names

- class name, creating instance variables with, 117-119
- method names, 144-145

- superclass name, creating instance variables with, 119-123

#### naming classes, 98-99

#### nesting messages, 36

#### NeXT, 7

#### nextObject method, 214

#### NeXTSTEP, 7

#### nil value, 212

#### notifications, 182-186

- posting, 185
- registering to receive, 186
- removing observers, 186
- values, 183-185

#### NSArray class, 209-210

- creating and enumerating, 215-217
- enumerateObjectsUsingBlock: method, 268
- iterating through, 206

#### NSAutoreleasePool class, 228

#### NSDate class, 208

#### NSDictionary class

- enumerateKeysAndObjectsUsingBlock: method, 268
- overview, 209-210

#### NSEnumerationConcurrent class, 266

#### NSEnumerationReverse class, 267

#### NSEnumerator class, 256

- methods, 214-215
- NSEnumerator instances, creating, 215

#### NSError class, 274-275

#### NSException class, 272-274

#### NSInvocation object

- creating, 172
- invoking, 175
- overview, 172
- properties, 173-175

#### NSLog class, 295-297

#### NSLogv class, 295

#### NSNotification class, 183

#### NSNotificationCenter class, 185

#### NSNumber class

- getters, 192
- overview, 183, 208

#### NSObject class, 34, 179-180

#### NSOperationQueue class, 287

#### NSProxy class, 180

#### NSSet class

- enumerateObjectsUsingBlock: method, 268
- overview, 209-210

#### NSString class

- enumerateLinesUsingBlock: method, 267-268
- overview, 209

#### NSStringFromClass, 296

#### NSStringFromSelector class, 296

#### numberWithBool method, 142

## O

objc\_getAssociatedObject function, 251-252

objc\_removeAssociatedObjects function, 251

objc\_setAssociatedObject function, 250, 251, 254



*Object Oriented Programming: An Evolutionary Approach (Cox)*, 179

Object value (NSNotification), 184

objectAtIndex method, 218

objectEnumerator method, 215

objectForKey method, 215, 218

object-oriented programming

accessors, 28

classes, 25

data abstraction, 24-25

encapsulation, 25-28

frameworks

layers, 23-24

overview, 22

implementing, 21-22

inheritance, 29-30

instances, 25

messages

nesting, 36

overview, 33

simple example, 33-34

methods

class methods, 35

declaring, 34

parameters, 36-37

signatures, 36-37

objects

allocating, 37-38

initializing, 37-38

overview, 21

projects, 22

variables

strongly typed variables,  
35-36

weakly typed variables,  
35-36

objects. *See also* classes

allocating, 37-38

associative references,  
250-251

adding, 251

getting and setting, 252

hiding in declared property,  
253

removing, 254

base objects, overriding, 29

collections

accessing objects in, 218

advantages of, 205-207

arrays, creating, 215-217

arrays, reading and writing,  
213

building from property lists  
at runtime, 209

collection classes,  
209-210

creating with literals, 212

creation methods,  
210-212

dictionaries, reading and  
writing, 213-214

enumeration, 214-217

mutable collections, 207

optimization, 211

overview, 205-207

in property lists, 207-209

sets, creating, 214

testing membership in,  
217

debugging, 104

delegates

Master-Detail Application  
template example,  
232-235

overview, 233

distributed objects, 182

initializing, 37-38

memory management. *See*  
memory management

NSInvocation

creating, 172

invoking, 175

overview, 172

properties, 173-175

receivers, 166

selectors

creating from string,  
169-170

creating with @selector,  
169

implementing, 170-171

with Interface Builder, 171  
overview, 166

performSelector, 169-171

SEL data type, 168-169

setting, 174

testing whether instance  
can respond to selector,  
175-176

observers, removing, 186

operating system services, 181

operation queues, 287-288

optimization for collections, 211

@optional directive, 236, 246

optional methods (protocols), 236

**OS X projects**

- actions
  - building projects with, 151-154
  - OS X versus iOS, 159-161
- project files, 65-66
- structure of, 64-65
- testbed apps
  - adding text fields, 82-88
  - building, 78-81
  - connecting text fields to code, 88
  - message syntax, 94-95
  - sending messages to text fields, 92-94

**overriding base objects, 29****P****packages, definition of, 317****packaging apps, 321-324**

- iOS archives, 324
- Mac archives, 322-323

**paradigms, 149****parameters (method), 36-37****pasting method declarations, 143****.pch files (precompiled header files), 66, 319****performSelector, 169-171****personal membership (developer program), 9-10****placing class instances in context, 112****plist files, 66, 319****plus sign (+), 35, 142****pointers, 314**

- dangling pointers, 227
- inline function pointers, 261
- typedef function pointer, 261-262

**posting notifications, 185****postNotification method, 185****postNotificationName method, 185****precompiled header files (.pch), 66, 319****predicates, 181****prefix headers, 67-69****\_\_PRETTY\_FUNCTION\_\_ macro, 295****primitive types, 205****private variables, 123****project info, 12****projects**

- creating, 16, 43-45
- files
  - .h files, 70-71
  - .m files, 71-73
  - main.m file, 69-70
  - prefix headers, 67-69
- iOS projects, 64
- object-oriented projects, 22
- OS X projects, 64-65
- project files, 65-66
- project info, 12
- source code control
  - branches, 47-48, 50-56
  - Git, 50-56
  - overview, 42-47
  - repository tips, 48-49

## structure of, 63

- targets, 12
- workspace, 12-16

**properties**

- accessing
  - with dot syntax, 134-135
  - with message syntax, 133
- adding to code, 133
- attributes, 133, 135

## accessor methods, 136

## atomicity, 137

## other attribute decorators,

137

## setter semantics, 137

## writability, 136

## compared to instance

variables, 127

## declared properties

- adding, 201
- attributes, 226-227, 253

## declaring

attributes, 133

example, 130-133

memory management, 132

## implementing, 138-139

NSInvocation object, 173-175

## property lists

building collections from at runtime, 209

collections in, 207-209

writing dictionaries as, 213

strong, 133

**property lists**

building collections from at runtime, 209

collections in, 207-209

writing dictionaries as, 213

protected variables, 123

@protocol directive, 74, 108

protocols, 30

    compared to categories, 239

    informal protocols, 246

    Master-Detail Application  
    template example, 232-235

    optional methods, 236

    overview, 106, 232-233

    turning classes into, 181

public variables, 123

## Q

qualifiers, variable qualifiers, 227

queues

    dispatch queues, 287

        adding asynchronous tasks  
        to, 289

    designing with, 290

    explained, 287

    global concurrent dispatch  
    queues, 288-289

    dispatch sources, 287

    operation queues, 287-288

    overview, 286

## R

reading

    arrays, 213

    dictionaries, 213-214

reason property (NSException),  
274

receivers, 166

receiving notifications, 186

reference counting

    ARC (automatic reference  
    counting)

        declared property  
        attributes, 226-227

        explained, 113, 225-227

    explained, 222

    manual reference counting,  
    222-225

references

    associative references,  
    250-251

        adding, 251

        getting and setting, 252

        hiding in declared  
        property, 253

        removing, 254

    forward references, 107-108

    reference counting

        ARC (automatic reference  
        counting), 113

        explained, 222

        manual reference counting,  
        222-225

registering

    for Apple IDs, 10

    for notifications, 186

release messages, 224, 228

remote repositories, 58-60

removeObserver method, 186

removing

    actions, 161

    associative references, 254

    observers, 186

repeat statements, 316

repositories

    checking out, 60

    cloning, 59

    remote repositories, 58-60

    tips, 48-49

@required directive, 236

respondToSelector method, 176

retain messages, 224, 228

return values

    getting, 174-175

    overview, 143-144, 146-147

returning multiple result values

    from methods, 112-114

reverseObjectEnumerator  
method, 215

Rhapsody, 7

Ritchie, Dennis M., 24

root classes (Foundation  
framework), 180

.rtf files, 65

runtime, 167-168

## S

Safari developer program, 9

*Sams Teach Yourself C in 21 Days*  
(Jones and Aitkin), 313

scripting methods, 182

SEL data type, 168-169

@selector directive, 169

selectors

    creating from string, 169-170

    creating with @selector, 169

    implementing, 170-171

- with Interface Builder, 171
  - overview, 166
  - performSelector, 169-171
  - SEL data type, 168-169
  - setting, 174
  - testing whether instance
    - can respond to selector, 175-176
  - sending messages to text fields, 92-94**
  - setArgument method, 174**
  - setByAddingObject method, 214**
  - setByAddingObjectsFromArray method, 214**
  - setByAddingObjectsFromSet method, 214**
  - sets**
    - accessing objects in, 218
    - creating, 214
    - testing membership in, 217
  - setSelector method, 174**
  - setTarget method, 173**
  - setter semantics, 137, 226**
  - setWithArray method, 214**
  - setWithObjects method, 215**
  - setWithSet method, 210**
  - signatures (method)**
    - creating, 172
    - overview, 36-37, 172
  - Simula, 24**
  - small-scale enumeration, 267**
  - Smalltalk, 24, 166**
  - source code control**
    - branches
      - creating, 50-56
      - overview, 47-48
    - Git, 50-56
      - discarding changes, 56
      - history of, 48
      - tracking changes, 52-55
    - overview, 42-47
    - remote repositories, 58-60
    - repository tips, 48-49
    - Subversion, 58
  - square brackets ([ ]), 94**
  - statements**
    - if, 315
    - repeat, 316
    - switch, 315
  - static typing, 117-122**
  - step-by-step conversion routine, building, 194-200**
  - strings, 169-170**
    - .strings files, 65
  - stringWithUTF8String, 296**
  - strong keyword, 226**
  - strong property, 133**
  - \_\_strong— variable qualifier, 227**
  - strongly typed variables, 35-36**
  - struct type, 314**
  - subclasses**
    - compared to categories, 241-242
    - pros and cons, 231-232
  - Subversion**
    - remote repositories, 58-60
    - on single computer, 58
  - superclass methods, calling, 104**
  - superclass name, creating instance variables with, 119-123**
  - switch statements, 315**
  - at symbol (@), 74**
  - synchronizing with locks, 287**
  - @synthesize directive, 130-132, 138**
- ## T
- targets**
    - overview, 12
    - setting, 173
  - tasks, adding to queues, 289**
  - testbed apps**
    - building on iOS, 78
    - building on OS X, 79
    - creating, 193-196
    - message syntax, 94-95
    - overview, 77-78
    - text fields, 81
      - adding in iOS, 82-84
      - adding in OS X, 85-87
      - connecting to code in iOS, 89-91
      - connecting to code in OS X, 91
      - sending messages to, 92-94
  - testing**
    - membership in collections, 217
    - whether instance can respond to selector, 175-176
  - text fields, 78**
    - adding in iOS, 82-84
    - adding in OS X, 85-87

- connecting to code in iOS, 89-91
- connecting to code in OS X, 91
- sending messages to, 92-94
- threading methods, 182**
- threads, concurrency, 284-285**
- throwing exceptions, 278-279**
- time-saving features**
  - associative references, 250-251
    - adding, 251
    - getting and setting, 252
    - hiding in declared property, 253
    - removing, 254
  - fast enumeration, 254-256
    - with NSEnumerator, 256
    - without NSEnumerator, 255
  - overview, 249-250
- Torvalds, Linus, 48**
- tracking changes with Git, 52-55**
- troubleshooting. See also debugging**
  - errors
    - in Cocoa and Cocoa Touch, 277
    - compared to exceptions, 272
    - NSError class, 274-275
    - overview, 271-272
  - exceptions
    - catching, 280
    - compared to errors, 272
    - identifying, 275-278
  - NSError class, 272-274
  - throwing, 278-279
  - @try, 275-278**
  - try/catch blocks, 275-278**
  - typedef function pointer, 261-262**
  - types, 313**
    - arrays
      - iterating through, 206
      - overview, 315
    - dynamic typing, 190
    - enumerated type, 313
    - id, 114-117, 129
    - pointers, 314
    - primitive types, 205
    - SEL, 168-169
    - struct, 314

## U

- UIKit, 179**
- university team membership (developer program), 10**
- unsafe\_unretained keyword, 227**
- \_\_unsafe\_unretained— variable qualifier, 227**
- URLs**
  - creating dictionaries from, 214
  - URL classes, 182
  - writing arrays to, 213
  - writing dictionaries to, 214
- userInfo property**
  - NSError class, 274
  - NSError class, 274

- userInfo value (NSNotification), 185**
- useTypeDefFunctionPointer method, 262**

## V

- value objects, 180**
- variable qualifiers, 227**
- variables**
  - autoreleasing variables, 228
  - block variables, 264-265
  - declaring
    - basic variable declaration, 128-130
    - legacy instance variable declarations, 138
  - id, 35
  - instance variables. *See also*
    - associative references
    - compared to properties, 127
    - creating for
      - CurrencyConverter, 114-117
    - creating for
      - CurrencyConverter with class name, 117-119
    - creating for
      - CurrencyConverter with superclass name, 119-123
    - declaring, 107, 111
    - visibility, 123
  - strongly typed variables, 35-36

- variable qualifiers, 227
  - weakly typed variables, 35-36
  - versions of Xcode, 41-42**
  - viewing console logs, 294**
  - views (Cocoa), 77**
  - visibility of instance variables, 123**
- ## W
- weak keyword, 226**
  - \_\_weak— variable qualifier, 227**
  - weakly typed variables, 35-36**
  - windows (Cocoa), 77**
  - workspace, 12-16**
  - writability, 136**
  - writeToFile method, 213**
  - writeToURL method, 213-214**
  - writing**
    - arrays, 213
    - dictionaries, 213-214
    - method declarations, 146
    - project files, 65-66
    - project info, 12
    - structure of, 63
  - source code control**
    - branches, 47-48, 50-57
    - Git, 50-56
    - overview, 42-47
    - remote repositories, 58-60
    - repository tips, 48-49
    - Subversion, 58
  - targets, 12
  - versions, 41-42
  - workspace, 12-16
- Xerox PARC, 165**
  - .xib files, 65**
  - XML support classes, 181**

## X-Y-Z

### Xcode

- classes, creating, 99-103
- downloading, 41
- overview, 11, 41-42
- projects, 41-42
  - creating, 17, 43-45
  - iOS projects, 64
  - OS X projects, 64-65