Updated for iOS 5 and Lion

Learn the programming language behind iPhone, iPad, and Mac apps development

Objective-C for Absolute Beginners

iPhone, iPad, and Mac Programming Made Easy

SECOND EDITION

Gary Bennett | Mitch Fisher | Brad Lees

Free XCELME TRAINING WEBINARS

Apress®

Objective-C for Absolute Beginners

iPhone, iPad, and Mac Programming Made Easy

Second Editon

Gary Bennett

Mitch Fisher

Brad Lees

Apress®

Objective-C for Absolute Beginners: iPhone, iPad, and Mac Programming Made Easy, Second Edition

ISBN-13 (pbk): 978-1-4302-3653-5

ISBN-13 (electronic): 978-1-4302-3654-2

Distributed to the book trade worldwide by Springer Science+Business Media, LLC., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail `orders-ny@springer-sbm.com`, or visit `www.springeronline.com`.

For information on translations, please e-mail `rights@apress.com` or visit `www.apress.com`.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales–eBook Licensing web page at `www.apress.com/bulk-sales`.

Any source code or other supplementary materials referenced by the author in this text is available to readers at `www.apress.com`. For detailed information about how to locate your book’s source code, go to `http://www.apress.com/source-code/`.

I'd like to dedicate this book to the two people who had the biggest impact on my career: my dad, Don W. Bennett, and Steve Jobs. Both passed away this year. Thanks for inspiring me to work in the field where I can have fun, make a difference, be creative, and live the American Dream.

—Gary Bennett

I would like to thank my family and friends who have always supported my endeavors. I would especially like to thank Heather, Matthew, and my two children Eric and Jade for patiently living without me for the many nights and forever-busy weekends. I would also like to thank my friends Gary and Brad for all the help they provided. It's been great working with them again.

—Mitch Fisher

I would like to thank my wife Natalie and my kids for the support and time they have given me to work on this book. I am also grateful for good friends who persuade me to take on crazy endeavors.

—Brad Lees

Contents at a Glance

Contents

About the Authors

Gary Bennett is president of xcelMe.com. xcelMe.com provides iPhone/iPad programming courses online. Gary has taught thousands of students how to develop iPhone/iPad apps, and has several very popular apps in the iTunes App Store. Gary's students have some of the best-selling apps in the iTunes App Store. Gary also worked for 25 years in the technology and defense industries. He served 10 years in the US Navy as a nuclear engineer aboard two nuclear submarines. After leaving the Navy, Gary worked for several companies as a software developer, CIO, and president. As CIO, he helped take VistaCare public in 2002. Gary also coauthored *iPhone Cool Projects* for Apress. He lives in Scottsdale, Arizona, with his wife Stefanie and their four children.

Mitch Fisher is a software developer in the Phoenix, Arizona, area. He was introduced to PCs back in the 1980s when 640 KB was considered more memory than anyone could ever use. Over the last 25 years, Mitch has worked for several large and medium-sized companies as a software engineer, software architect, and software manager, and has led teams of developers on multimillion-dollar projects. Mitch now divides his time between writing iOS applications, creating server-side UNIX technologies, and teaching iOS development at `xcelMe.com`.

Brad Lees has more than 14 years of experience in application development and server management. He has specialized in creating and initiating software programs in real estate development systems and financial institutions. Highlights of his professional career include his positions as information systems manager at The Lyle Anderson Company, product development manager for Smarsh, vice president of application development for iNation, and information technology manager at The Orcutt/Winslow Partnship, the largest architectural firm in Arizona. A graduate of Arizona State University, Brad resides in Phoenix with his wife Natalie and their five children.

About the Technical Reviewer

James Bucanek has spent the past 30 years programming and developing microcomputer systems. He has experience with a broad range of technologies, from embedded consumer products to industrial robotics. James currently focuses on Macintosh and iPhone software development. When not programming, James indulges in his love of the arts. He earned an associate degree in classical ballet from the Royal Academy of Dance, and occasionally teaches at Adams Ballet Academy.

Acknowledgments

We would like to thank Apress for all their help in making this book possible. Specifically, we would like to thank Kelly Moritz, our coordinating editor, for helping us stay focused and overcoming many obstacles. Without Kelly, this book would not have been possible.

Special thanks to Matthew Moodie, our development editor, for all his suggestions during the editorial review process to help make this a great book. Thanks to Chandra Clarke and Scribendi, Inc., the copy editors who made the book look great.

We would also like to thank the Alice Community and Carnegie Mellon University for developing Alice and making learning object-oriented programming fun and easy!

Introduction

Over the last three years, we've heard the following countless times:

- "I've never programmed before, but I have a great idea for an iPhone/iPad app."
- "Can I really learn to program the iPhone or iPad?"

We always answer, "Yes, but you have to believe you can." Only you are going to tell yourself you can't do it.

For the Newbie

This book assumes you may have never programmed before. The book is also written for someone who may have never programmed before using object-oriented programming (OOP) languages. There are many Objective-C books out there, but all of these books assume you have programmed before and know OOP and computer logic. We wanted to write a book that takes readers from knowing little or nothing about computer programming and logic to being able to program in Objective-C. After all, Objective-C is the native programming language for the iPhone, iPad, and Mac.

Over the last three years, we have taught well over a thousand students at xcelMe.com to be iPhone/iPad (iOS) developers. Many of our students have developed some of the most successful iOS apps in their category in the iTunes App Store. We have incorporated what we have learned in our first two courses, Introduction to Object-oriented Programming and Logic and Objective-C for iPhone/iPad Developers, into this book.

For the More Experienced

Many developers who programmed years ago or programmed in a non-OOP language need a background in OOP and Logic before they dive into Objective-C. This book is for you. We gently walk you through OOP and how it is used in iOS development to help make you a successful iOS developer.

Why Alice: An Innovative 3D Programming Environment

Over the years, universities have struggled with several issues with their computer science departments:

- High male-to-female ratios
- High drop-out rates

- Longer than average time to graduation

One of the biggest challenges to learning OOP languages like Java, C++, or Objective-C is the steep learning curve from the very beginning. In the past, students had to learn the following topics all at once:

- Object-oriented principles
- A complex Integrated Development Environment (IDE), i.e., Xcode, Eclipse, Visual Studio
- The syntax of the programming language
- Programming logic and principles

As a result, Carnegie Mellon University received a grant from the US government and developed Alice. Alice, an innovative 3D programming environment, makes it easy for new developers to create rich graphical applications. Alice is a teaching tool for students learning to program in an OOP environment. The software uses 3D graphics and a drag-and-drop interface to facilitate a more engaging, less frustrating first programming experience.

Alice enables students to focus on learning the principles of OOP without having to focus on learning a complex IDE and Objective-C principles all at once. You get to focus on each topic individually. This helps students feel a real sense of accomplishment as they progress.

As drag-and-drop programming, Alice removes all the complexity of learning an IDE and programming language syntax. You'll see programming is actually fun, and you can develop very cool and sophisticated apps in Alice.

After we introduce the OOP topic and readers feel comfortable with the material, we then move into Xcode, where you get to use your new OOP knowledge in writing Objective-C applications. This way, you can focus on the Objective-C syntax and language without having to learn OOP at the same time.

Learning Objective-C Without Alice

More than a thousand `xcelMe.com` students have used this book to become successful iOS developers. At the end of each course, we ask our students if the Alice sections in the first four sections were useful. More than half of the students thought using Alice at the beginning of the first four chapters to introduce the chapter was critical to their success. However, some of the students didn't feel they needed the Alice examples at the beginning of the first four chapters.

We have laid out the first four chapters of this book with the first part of each chapter introducing the OOP topic with Alice; the remaining part of the chapter introduces the topic using Objective-C. Thus, you can skip the Alice material if you feel comfortable with the topic.

How This Book Is Organized

You'll notice that we are all about successes in this book. We introduce the OOP and Logic concepts in Alice and then move those concepts to Xcode and Objective-C. Many students are visual or learn by doing. We use both techniques. We'll walk you through topics and concepts with visual examples and then take you through step-by-step examples reinforcing the concepts.

We often repeat topics in different chapters to reinforce what you have learned and apply these skills in new ways. This enables new programmers to reapply development skills and feel a sense of accomplishment as they progress. Don't worry if you feel you haven't mastered a topic. Keep moving forward!

The Formula for Success

Learning to program is an interactive process between your program and you. Just like learning to play an instrument, you have to practice. You must work through the examples and exercises in this book. Understanding the concept doesn't mean you know how to apply it and use it.

You will learn a lot from this book. You will learn a lot from working through the exercises in this book. *However, you will really learn when you debug your programs.* Spending time walking through your code and trying to find out why it is not working the way you want is an unparalleled learning process. The downside of debugging is a new developer can find it especially frustrating. If you have never wanted to throw your computer out the window, you will. You will question why you are doing this, and whether you are smart enough to solve the problem. Programming is very humbling, even for the most experienced developer.

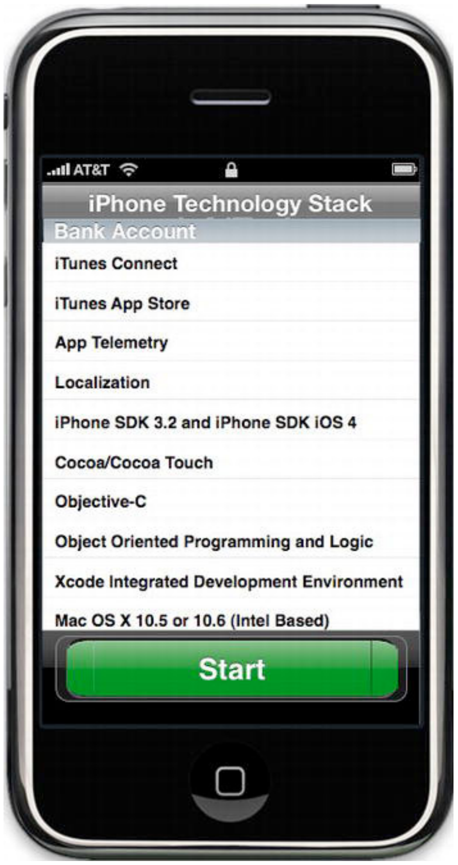
Like a musician, the more you practice the better you get. By practicing, we mean programming! You can do some amazing things as a programmer. The world is your oyster. Seeing your app in the iTunes App Store is one of the most satisfying accomplishments. However, there is a price, and that price is time spent coding and learning.

Having taught more than a thousand students to become iOS developers, we have put together a formula for what makes students successful. Here is our formula for success:

- Believe you can do it. You'll be the only one who says you can't do this. So don't tell yourself that.
- Work through all the examples and exercises in this book.
- Code, code, and keeping coding. The more you code, the better you'll get.
- Be patient with yourself. If you were fortunate enough to have been a 4.0 student who can memorize material just by reading it, this will not happen with Objective-C coding. You are going to have to spend time coding.
- You learn by reading this book. You really learn by debugging your code.
- Use the ***free*** xcelMe.com webinars and YouTube videos explained at the end of this chapter.
- Don't give up!

The Development Technology Stack

We will walk you through the process of understanding the development process for your iOS apps and what technology you need. However, briefly looking at all the pieces together is helpful. For a sample iPhone app in a Table View, see Figure 1.



iPhone Technology Stack

Bank Account

iTunes Connect

iTunes App Store

App Telemetry

Localization

iPhone SDK 3.2 and iPhone SDK iOS 4

Cocoa/Cocoa Touch

Objective-C

Object Oriented Programming and Logic

Xcode Integrated Development Environment

Mac OS X 10.5 or 10.6 (Intel Based)

Figure 1. *The iPhone/iPad technology stack*

Required Software, Materials, and Equipment

One of the great things about Alice is it available on the three main operating systems used today:

- Windows
- Mac
- Linux

The other great thing about Alice is it is free! You can download Alice at `www.Alice.org`.

Operating System and IDE

Although you can use Alice on many platforms, the Integrated Development Environment (IDE) that developers use to develop iOS apps is Xcode. You have to use *an Intel-based Mac to use Xcode and submit apps!* Xcode is free and is available in the Mac App Store.

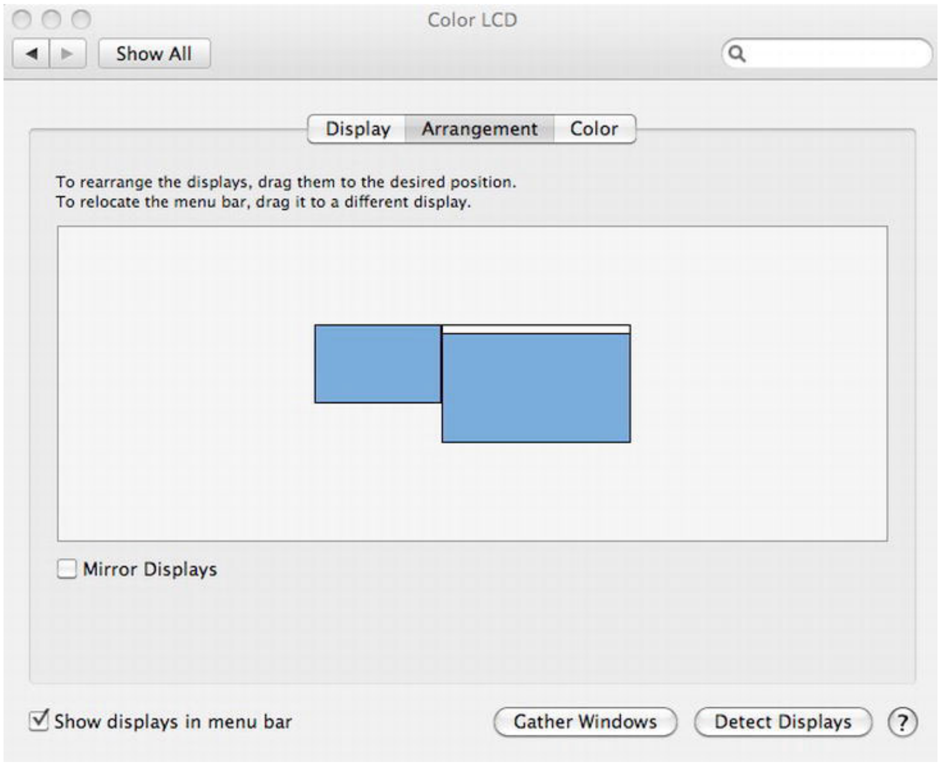
Software Development Kits

You will need to register as an iOS developer. You can do this at `http://developer.apple.com/iphone`.

When you are ready to upload your app to the iTunes App Store, you will need to pay $99/year.

Dual Monitors

We recommend developers have a second monitor connected to their computer. It is great to step through your code and watch your output window and iPad simulator at the same time on dual independent monitors. Apple hardware makes this easy. Just plug your second monitor into the display port of any Intel-based Mac, with the correct Mini DisplayPort adapter of course, and you have two monitors working independently of one another. See Figure 2. Note that dual monitors are not required. You will just have to organize your open windows to fit on your screen if you don't.



To rearrange the displays, drag them to the desired position.
To relocate the menu bar, drag it to a different display.

☐ Mirror Displays

☑ Show displays in menu bar

Figure 2. *Dual monitors*

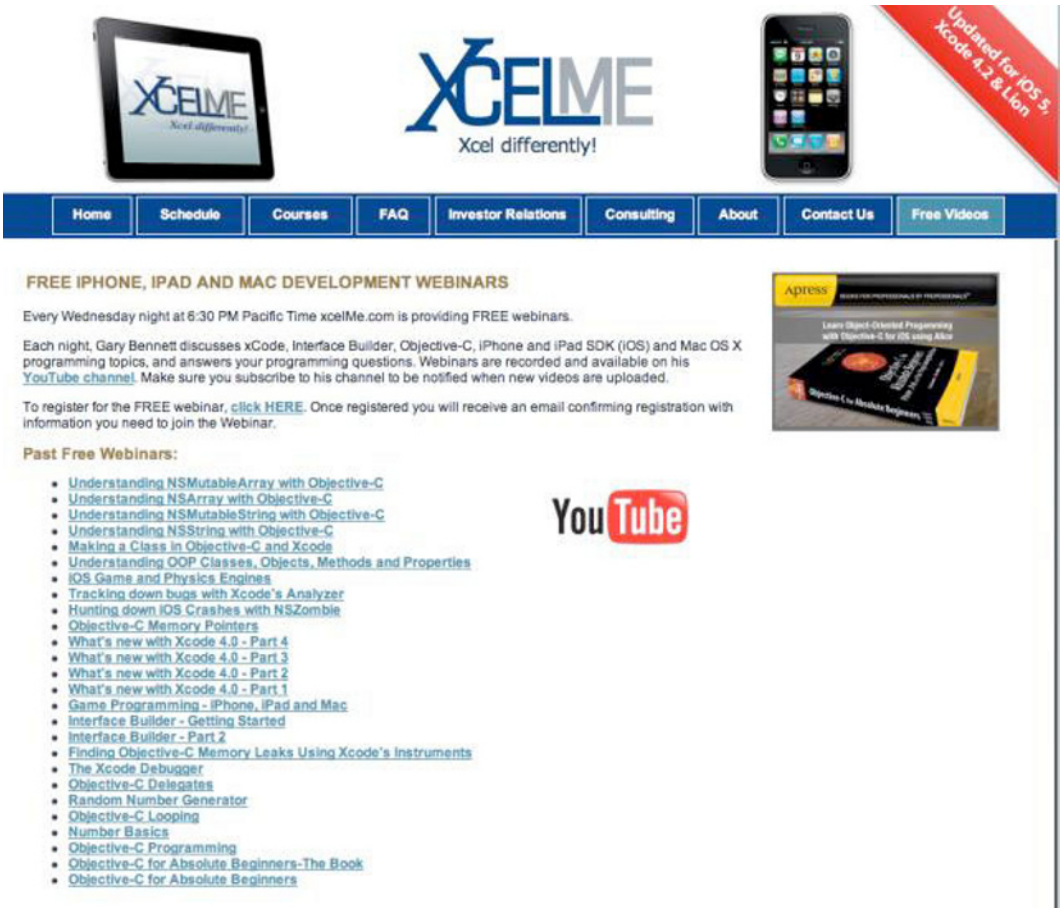
Free Live Webinars, Q&A, and YouTube Videos

Nearly every Wednesday night at 7:30 p.m. Pacific daylight time, we have live webinars and discuss a topic from the book or a timely item of interest. These webinars are free, and you can register for them at `www.xcelme.com/free-webinars.php`.

At the end of the webinars, we do a Q&A. You can ask a question on the topic discussed or any topic in the book.

Additionally, all these webinars are recorded and available on YouTube.

Make sure you subscribe to the YouTube channel so you are notified when new recordings are uploaded.



FREE IPHONE, IPAD AND MAC DEVELOPMENT WEBINARS

Every Wednesday night at 6:30 PM Pacific Time xcelMe.com is providing FREE webinars.

Each night, Gary Bennett discusses xCode, Interface Builder, Objective-C, iPhone and iPad SDK (iOS) and Mac OS X programming topics, and answers your programming questions. Webinars are recorded and available on his YouTube channel. Make sure you subscribe to his channel to be notified when new videos are uploaded.

To register for the FREE webinar, click HERE. Once registered you will receive an email confirming registration with information you need to join the Webinar.

Past Free Webinars:

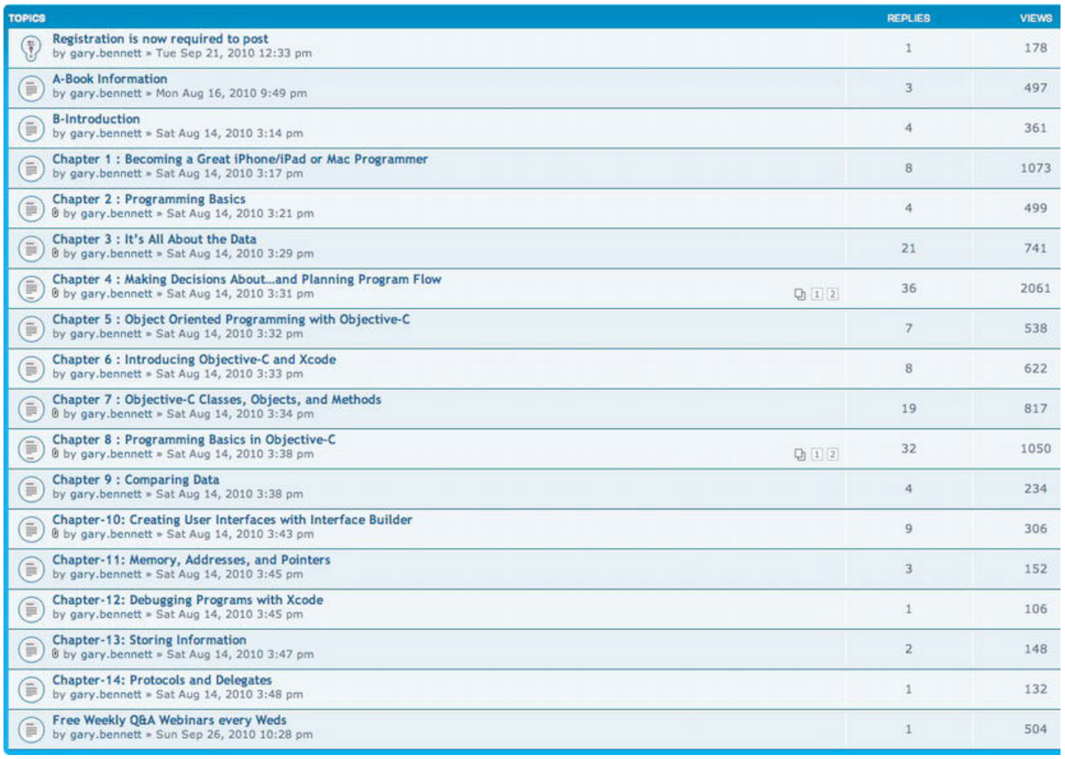
- Understanding NSMutableArray with Objective-C
- Understanding NSArray with Objective-C
- Understanding NSMutableString with Objective-C
- Understanding NSString with Objective-C
- Making a Class in Objective-C and Xcode
- Understanding OOP Classes, Objects, Methods and Properties
- iOS Game and Physics Engines
- Tracking down bugs with Xcode's Analyzer
- Hunting down iOS Crashes with NSZombie
- Objective-C Memory Pointers
- What's new with Xcode 4.0 - Part 4
- What's new with Xcode 4.0 - Part 3
- What's new with Xcode 4.0 - Part 2
- What's new with Xcode 4.0 - Part 1
- Game Programming - iPhone, iPad and Mac
- Interface Builder - Getting Started
- Interface Builder - Part 2
- Finding Objective-C Memory Leaks Using Xcode's Instruments
- The Xcode Debugger
- Objective-C Delegates
- Random Number Generator
- Objective-C Looping
- Number Basics
- Objective-C Programming
- Objective-C for Absolute Beginners-The Book
- Objective-C for Absolute Beginners

Figure 3. *Free Objective-C webinars and YouTube videos*

Free Book Forum

We have developed an online forum for this book at `http://forum.xcelme.com`, where you can ask questions while you are learning Objective-C and get answers from the authors. You will also find answers to the exercises and additional exercises to help you learn. See Figure 3.

You can also access answers to exercises and discover helpful links to help you become a successful iPhone/iPad developers and create great apps. See Figure 4. So let's get started!



TOPICS	REPLIES	VIEWS
Registration is now required to post by gary.bennett » Tue Sep 21, 2010 12:33 pm	1	178
A-Book Information by gary.bennett » Mon Aug 16, 2010 9:49 pm	3	497
B-Introduction by gary.bennett » Sat Aug 14, 2010 3:14 pm	4	361
Chapter 1 : Becoming a Great iPhone/iPad or Mac Programmer by gary.bennett » Sat Aug 14, 2010 3:17 pm	8	1073
Chapter 2 : Programming Basics by gary.bennett » Sat Aug 14, 2010 3:21 pm	4	499
Chapter 3 : It's All About the Data by gary.bennett » Sat Aug 14, 2010 3:29 pm	21	741
Chapter 4 : Making Decisions About...and Planning Program Flow by gary.bennett » Sat Aug 14, 2010 3:31 pm 1 2	36	2061
Chapter 5 : Object Oriented Programming with Objective-C by gary.bennett » Sat Aug 14, 2010 3:32 pm	7	538
Chapter 6 : Introducing Objective-C and Xcode by gary.bennett » Sat Aug 14, 2010 3:33 pm	8	622
Chapter 7 : Objective-C Classes, Objects, and Methods by gary.bennett » Sat Aug 14, 2010 3:34 pm	19	817
Chapter 8 : Programming Basics in Objective-C by gary.bennett » Sat Aug 14, 2010 3:38 pm 1 2	32	1050
Chapter 9 : Comparing Data by gary.bennett » Sat Aug 14, 2010 3:38 pm	4	234
Chapter-10: Creating User Interfaces with Interface Builder by gary.bennett » Sat Aug 14, 2010 3:43 pm	9	306
Chapter-11: Memory, Addresses, and Pointers by gary.bennett » Sat Aug 14, 2010 3:45 pm	3	152
Chapter-12: Debugging Programs with Xcode by gary.bennett » Sat Aug 14, 2010 3:45 pm	1	106
Chapter-13: Storing Information by gary.bennett » Sat Aug 14, 2010 3:47 pm	2	148
Chapter-14: Protocols and Delegates by gary.bennett » Sat Aug 14, 2010 3:48 pm	1	132
Free Weekly Q&A Webinars every Weds by gary.bennett » Sun Sep 26, 2010 10:28 pm	1	504

Figure 4. *Reader Forum for accessing answers to exercise and posting questions for authors*

Chapter 1

Becoming a Great iOS or Mac Programmer

Now that you're ready to become a software developer and have read the Introduction of this book, you need to become familiar with several key concepts. Your computer program will do exactly what you tell it to do—no more and no less. It will follow the programming rules that were defined by the operating system and programming language. Your program doesn't care if you are having a bad day or how many times you ask it to perform something. Often, what you think you've told your program to do and what it actually does are two different things.

KEY TO SUCCESS: If you haven't already, take a few minutes to read the Introduction of this book. The Introduction shows you where to go to access the free webinars, forums, and YouTube videos that go with each chapter. Also, you'll better understand why we are using the Alice programming environment and how to be successful in developing your iOS and Mac apps.

Depending on your background, working with something absolutely black and white may be frustrating. Many times, programming students have lamented, "That's not what I wanted it to do!" As you begin to gain experience and confidence programming, you'll begin to think like a programmer. You will understand software design and logic, and you will experience having your programs perform exactly as you want and the satisfaction associated with this.

Thinking like a Developer

Software development involves writing a computer program and then having a computer execute that program. A **computer program** is the set of instructions that we want the computer to perform. Before beginning to write a computer program, it is helpful to list the steps that we want our program to perform, in the order we want them accomplished. This step-by-step process is called an **algorithm**.

If we want to write a computer program to toast a piece of bread, we would first write an algorithm. This algorithm might look something like this:

1. Take the bread out of the bag.
2. Place the bread in the toaster.
3. Press the toast button.
4. Wait for the toast to pop up.
5. Remove the toast from the toaster.

At first glance, this algorithm seems to solve our problem. However, our algorithm leaves out many details and makes many assumptions. For example,

1. What kind of toast does the user want? Does the user want white bread, wheat, or some other kind of bread?
2. How does the user want the bread toasted? Light or dark?
3. What does the user want on the bread after it is toasted: butter, margarine, honey, or strawberry jam?
4. Does this algorithm work for all users in their cultures and languages? Some cultures may have another word for toast or not know what toast is.

Now, you might be thinking we are getting too detailed for just making a simple toast program. Over the years, software development has gained a reputation of taking too long, costing too much, and not being what the user wants. This reputation came to be because computer programmers often start writing their programs before they have really thought through their algorithms.

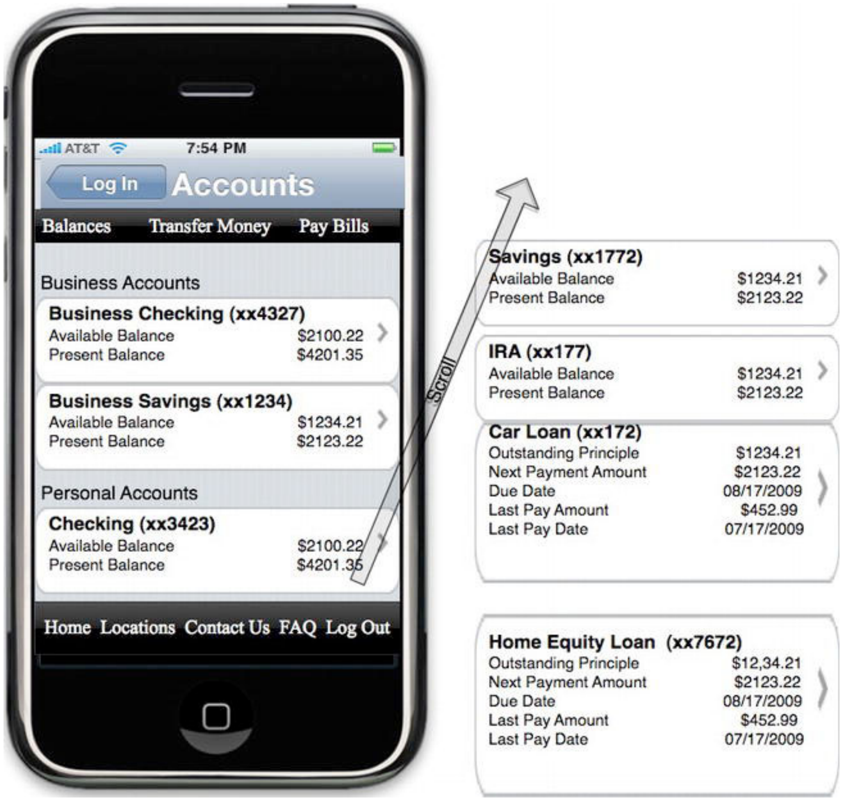
The key ingredients to making successful applications are **design requirements**. Design requirements can be very formal and detailed or as simple as a list on a piece of paper. Design requirements are important because they help the developer flush out what the application should do and not do when complete. Design requirements should not be completed in a programmer's vacuum, but should be produced as the result of collaboration between developers, users, and customers.

NOTE: If you take anything away from this chapter, take away the importance of considering design requirements and user interface design before starting software development. This is the most effective (and least expensive) use of time in the software development cycle. Using a pencil and eraser is a lot easier and faster than making changes to code because you didn't have others look at the designs before starting to program.

Another key ingredient to your successful app is the **user interface (UI)** design. Apple recommends that you spend over 50% of the entire development process focusing on the UI design. The design can be simple pencil-and-paper layouts created using the *iPhone Application Sketch Book or the iPad Application Sketch Book* by Dean Kaplan (Apress, 2009) or on-screen layout created with the Omni Group's OmniGraffle *(*`www.omnigroup.com`*)* software application with the Ultimate iPhone Stencil plug-in *(*`www.graffletopia.com`*)*. Many software developers start with the UI design, and after laying out all the screen elements and having many users look at paper mock-ups, they then write out the design requirements from their screen layouts.

After you have done your best to flush out all the design requirements, laid out all the user interface screens, and had the client(s) or potential customers look at your design and give you feedback, coding can begin. Once coding begins, design requirements and user interface screens can change, but the changes are typically minor and easily accommodated by the development process. See Figures 1–1 and 1–2.

Figure 1–1 shows a mock-up of a mobile banking app screen prior to development using OmniGraffle. Developing mock-up screens along with design requirements forces developers to think through many of the applications usability issues before coding begins. This enables the application development time to be shortened and makes for a better user experience and better reviews on the iTunes App Store. Figure 1–2 shows how the view for the mobile banking app actually appears when completed.



Business Accounts

Business Checking (xx4327)

Available Balance	$2100.22
Present Balance	$4201.35

Business Savings (xx1234)

Available Balance	$1234.21
Present Balance	$2123.22

Personal Accounts

Checking (xx3423)

Available Balance	$2100.22
Present Balance	$4201.35

Savings (xx1772)

Available Balance	$1234.21
Present Balance	$2123.22

IRA (xx177)

Available Balance	$1234.21
Present Balance	$2123.22

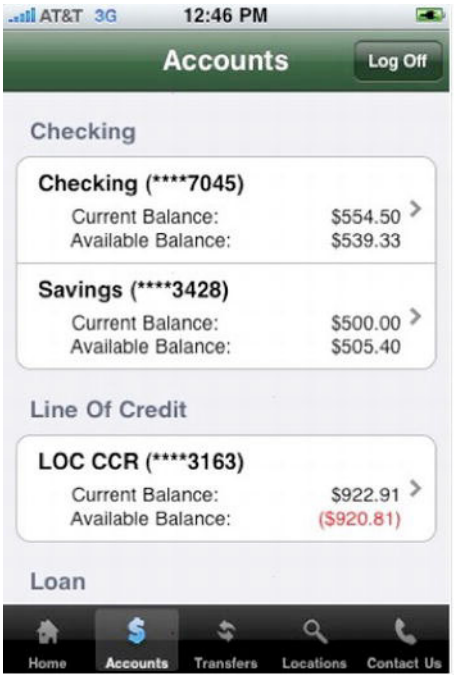
Car Loan (xx172)

Outstanding Principle	$1234.21
Next Payment Amount	$2123.22
Due Date	08/17/2009
Last Pay Amount	$452.99
Last Pay Date	07/17/2009

Home Equity Loan (xx7672)

Outstanding Principle	$12,34.21
Next Payment Amount	$2123.22
Due Date	08/17/2009
Last Pay Amount	$452.99
Last Pay Date	07/17/2009

Figure 1–1. *This is a User Interface (UI) mock-up of the Account Balance screen for an iPhone mobile banking app before development begins. This UI design mock-up was completed using OmniGraffle.*



Checking

Checking (****7045)
Current Balance: $554.50
Available Balance: $539.33

Savings (****3428)
Current Balance: $500.00
Available Balance: $505.40

Line Of Credit

LOC CCR (****3163)
Current Balance: $922.91
Available Balance: ($920.81)

Loan

Figure 1–2. *This screenshot shows a completed iPhone mobile banking application as it appeared on the iTunes App Store. This app is called Woodforest Mobile Banking.*

Completing the Development Cycle

Now that we have our design requirements and user interface designs and have written our program, what's next? After programming, we need to make sure our program matches the design requirements and user interface design and ensure that there are no errors. In programming vernacular, errors are called **bugs**. Bugs are undesired results of our programming and must be fixed before the app is released to the App Store. The process of finding bugs in programs and making sure the program meets the design requirements is called **testing**. Typically, someone who is experienced in software testing methodology and who didn't write the app performs this testing. Software testing is commonly referred to as **Quality Assurance (QA)**.

> **NOTE:** When an application is ready to be submitted to the iTunes App Store, Xcode gives the file an `.app` extension, for example, `appName.app`. That is why iPhone, iPad, and Mac applications are called **apps**. We will use "program," "application," and "app" to mean the same thing throughout this book.

During the testing phase, the developer will need to work with QA staff to determine why the application is not working as designed. The process is called **debugging**. It requires the developer to step through the program to find out why the application is not working as designed. Figure 1–3 shows the complete software development cycle.

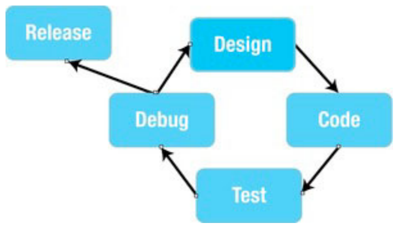


Figure 1–3. *The typical software development cycle*

Frequently during testing and debugging, changes to the requirements (design) must occur to make the application more usable for the customer. After the design requirements and user interface changes are made, the process begins over again.

At some point, the application that everyone has been working so hard on must be shipped to the iTunes App Store. Many considerations are taken into account when this happens:

- Cost of development
- Budget
- Stability of the application
- Return on investment

There is always the give-and-take between developers and management. Developers want the app perfect and management wants to start realizing revenue from the investment as soon as possible. If the release date were left up to the developers, the app would likely never ship to the App Store. Developers would continue to tweak the app forever, making it faster, more efficient, and more usable. At some point, however, the code needs to be pried from the developers' hands and uploaded to the App Store so it can do what it was meant to do.

Introducing Object Oriented Programming

As discussed in detail in the Introduction, Alice enables us to focus on **object oriented programming (OOP)** without having to cover all the Objective-C programming syntax and complex Xcode development environment in one big step. Instead, we can focus on learning the basic principles of OOP and using those principles quickly to write our first programs.

For decades, developers have been trying to figure out a better way to develop code that is reusable, manageable, and easily maintained over the life of a project. OOP was designed to help achieve code reuse and maintainability while reducing the cost of software development.

OOP can be viewed as a collection of objects in a program. Actions are performed on these objects to accomplish the design requirements.

An **object** is anything that can be acted on. For example, an airplane, person, or screen/view on the iPad can all be objects. We may want to act on the plane by making the plane bank. We may want the person to walk or to change the color of the screen of an app on the iPad. Actions are all being applied to these objects; see Figure 1–4.

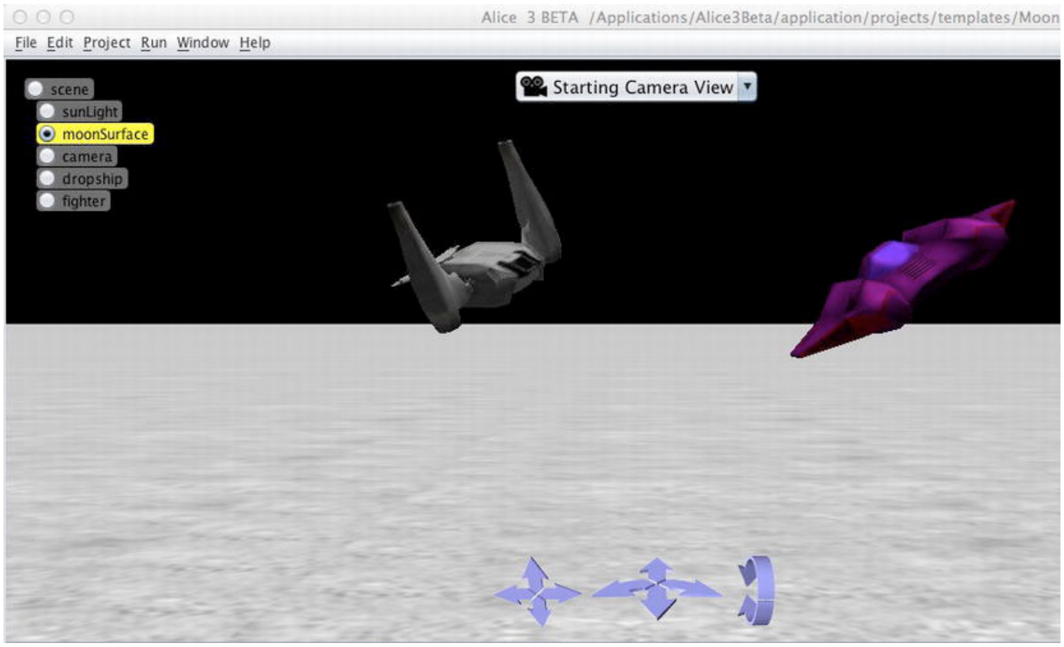

Alice 3 BETA /Applications/Alice3Beta/application/projects/templates/Moon
File Edit Project Run Window Help
Starting Camera View
scene
sunLight
moonSurface
camera
dropship
fighter

Figure 1–4. *These are two objects in an Alice application, a Dropship and Fighter. Both objects can have actions applied—takeoff and landing, turn right and turn left..*

Alice will run a program, such as the one shown in Figure 1–4, for you if you click the play button. When we run our Alice applications, the user can apply actions to the objects in our application. Similarly, Xcode is an **Integrated Development Environment (IDE)** that enables us to run our application from within our programming environment. We can test our applications on our computers first before running them on our iOS devices by running the apps in Xcode's simulator, as shown in Figure 1–5.

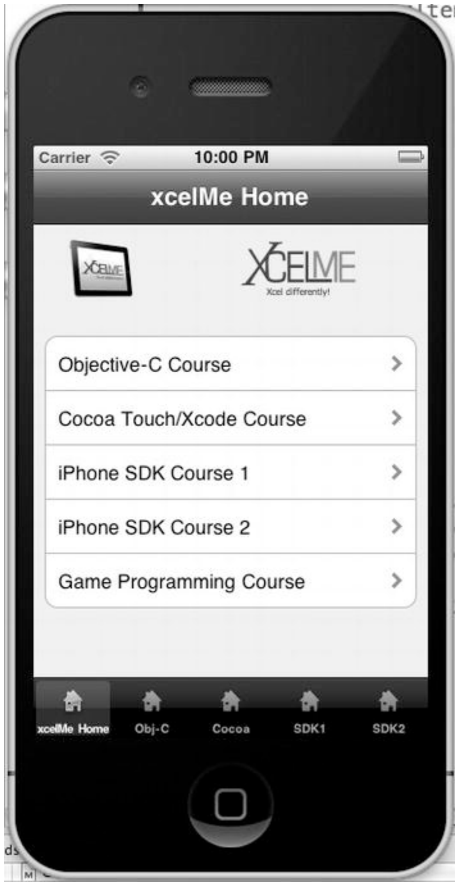


Figure 1–5. *This sample iPhone app contains a table object to organize a list of courses. Actions such as "rotate left" or "user did select row 3" can be applied to this object.*

Actions that are performed on objects are called **methods**. Methods manipulate objects to accomplish what we want our app to do. For example, for our jet object in Figure 1–4, we might have the following methods:

```
goUp
goDown
bankLeft
turnOnAfterBurners
lowerLandingGear
```

Our table object in Figure 1–5 is actually called `UITableView` when we use it in a program, and it could have the following methods:

```
loadView
shouldAutorotateToInterfaceOrientation
numberOfSectionsInTableView
cellForRowAtIndexPath
didSelectRowAtIndexPath
```

All objects have data that describes those objects. This data is defined as **properties.** Each property describes the associated object in a specific way. For example, the jet object's properties might be as follows:

```
altitude = 10,000 feet
heading = North
speed = 500 knots
pitch = 10 degrees
yaw = 20 degrees
latitude = 33.575776
longitude = -111.875766
```

For our `UITableView` object in Figure 1–5, the following might be our properties:

```
backGroundColor = Red
selectedRow = 3
animateView = No
```

An object's properties can be changed at any time when our program is running, when the user interacts with the app, or when the programmer designs the app to accomplish the design requirements. The values stored in the properties of an object at a specific time are collectively called the **state of an object**.

State is an important concept in computer programming. When teaching students about state, we ask them to go over to a window and find an airplane in the sky. We then ask them to snap their fingers and make up some of the values that the plane's properties might have at that specific time. Those values might be

```
altitude = 10,000 feet
latitude = 33.575776
longitude = -111.875766
```

Those values represent the state of the object at the specific time that they snapped their fingers.

After waiting a couple minutes, we ask the students to find that same plane, snap their fingers again, and record the plane's possible state at that specific point in time.

The values of the properties might then be something like

```
altitude = 10,500 feet
latitude = 33.575665
longitude = -111.875777
```

Notice how the state of the object changes over time.

Working with the Alice Interface

Alice offers a great approach in using the concepts that we have just discussed without all the complexity of learning Xcode and the Objective-C language at the same time. It takes only a few minutes to familiarize oneself with the Alice interface and begin writing a program.

The Introduction of this book describes how to download Alice. After it's downloaded and installed, you need to open Alice. It will look like Figure 1–6.

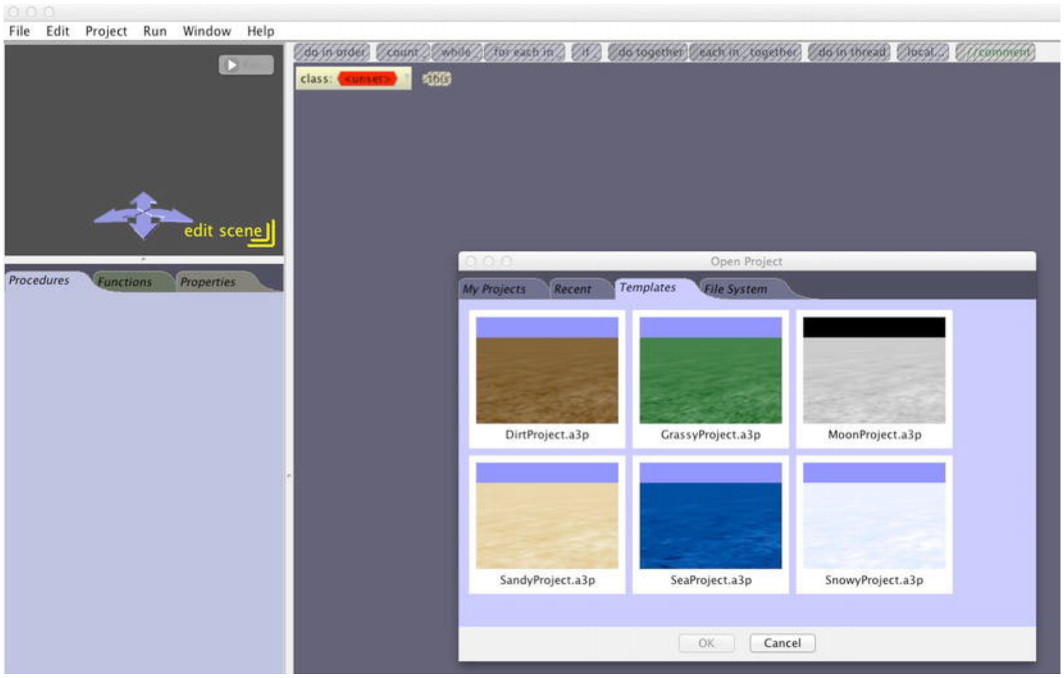


Figure 1–6. *Alice IDE running*

Technically speaking, Alice is not a true IDE like Xcode, but it is pretty close and much easier to learn than Xcode. A true IDE combines code development, user interface layout, debugging tools, documentation, and simulator/console launching for a single application; see Figure 1–7. However, Alice offers a similar look, feel, and features to Xcode. This will serve you well later when we start writing Objective-C code.