appendix iii

Tooltips

You've done a lot in this book.

Here's all the quick reference tips from the end of each chapter, collected together for you in one convenient place.

Your iPhone Toolbox

Here's all the tips you learned throughout the book, in one handy place.

Chapter 1: Getting started with iDecide

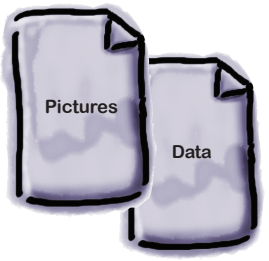
In this chapter, you got familiar with the **iPhone SDK** and added some basic **app interactions** to your toolbox.

Views are constructed in Interface Builder...

A view are made up of nib (*.xib) files and the GUIs are edited with Interface Builder

...then you write the code that makes the views work...

This code is almost always written in Objective-C using Xcode.



...and any other resources, all packaged into your application.

Images and other data are referenced together in Xcode so that all of the files that you need can be easily dealt with.

Chapter 2: Spinning up InstaTwit

In this chapter, you added the **picker control** and learned about **protocols**, **delegates**, and **datasources**.

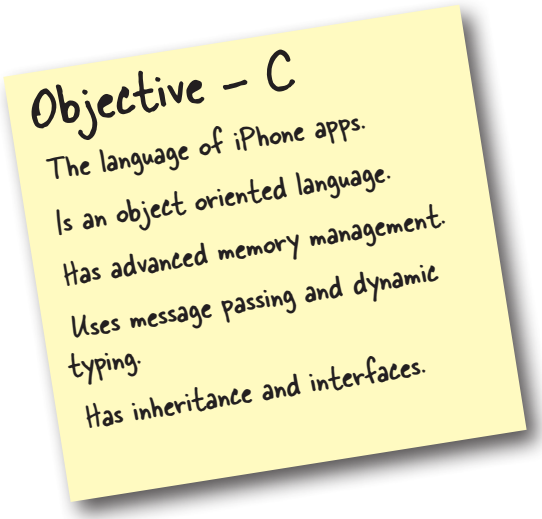
BULLET POINTS

- The picker needs a delegate and data source for it to work.
- In a picker, each dial is a component.
- In a picker, each item is a row.
- Protocols define the messages your class must realize - some of them might be optional.

Chapter 3: Getting to know Objective-C

In this chapter, you customized InstaTwit's **text options**, and got up close and personal with **Objective-C**.

Attribute	When you want it...
readwrite	When you want the property to be modifiable by people. The compiler will generate a getter and a setter for you. This is the default.
readonly	When you don't want people modifying the property. You can still change the field value backing the property, but the compiler won't generate a setter.
assign	When you're dealing with basic types, like ints, floats, etc. The compiler just creates a setter with a simple myField = value statement. This is the default, but not usually what you want.
retain	When you're dealing with object values. The compiler will retain the value you pass in (we'll talk more about retaining in a minute) and release the old value when a new one comes in.
copy	When you want to hold onto a copy of some value instead of the value itself. For example, if you want to hold onto an array and don't want people to be able to change its contents after they set it. This sends a copy message to the value passed in then retains that.



Chapter 4: Multiple views

In this chapter, you started working with **multiple views**, and found out what **navigation controllers** are all about.

Navigation Template

Comes with a table view and navigation control built in.

Is great for a productivity app.

Is designed to manage hierarchical data and multiple views.

Has cool animations built in to move between views.

Tables

Are a collection of cells.

Come with support for editing contents, scrolling, and moving rows.

A table can only have one column, but you can customize your cells to look like more than one column.

UITableView

Controls memory by only creating the cells requested in the view. Any other cells are destroyed if the iPhone needs the memory for something else.

Navigation controller

Maintains a view stack for moving between views.

Has a navigation bar for buttons and a title.

Can support custom toolbars at the bottom of the view as needed.

Plists

Arrays and Xcode can support plists.

Are a great way to store information.

Are good for handling data, but have some limitations – core data is another option with way fewer restrictions.

Debugging

Xcode has a built in console with debugging and logging information.

Gives you errors and warnings as you compile to identify problems.

The built in debugger allows you to set breakpoints and step through the code to find the bug.

Chapter 5: Plists and modal views

In this chapter, you **debugged** adding a plist of **dictionaries**, incorporated **modal views** to your app, and started navigating the process of getting your app approved for the **app store**.

Debugging

If you know where your problem likely is, then set the breakpoint there.

You can use the debugger to step through the problem area.

If you have no idea where to start, you can step through the entire app!

iTunes Basics

Submitting your app to the store means it HAS TO CONFORM TO THE HIG.

Approvals can take weeks, so try and get it right the first time..

Once your app is up for sale, the reviews stay with it, even with updates.

Dictionaries

Are useful ways to expand the contents of a plist.

Need to be properly handled inside the app.

Views

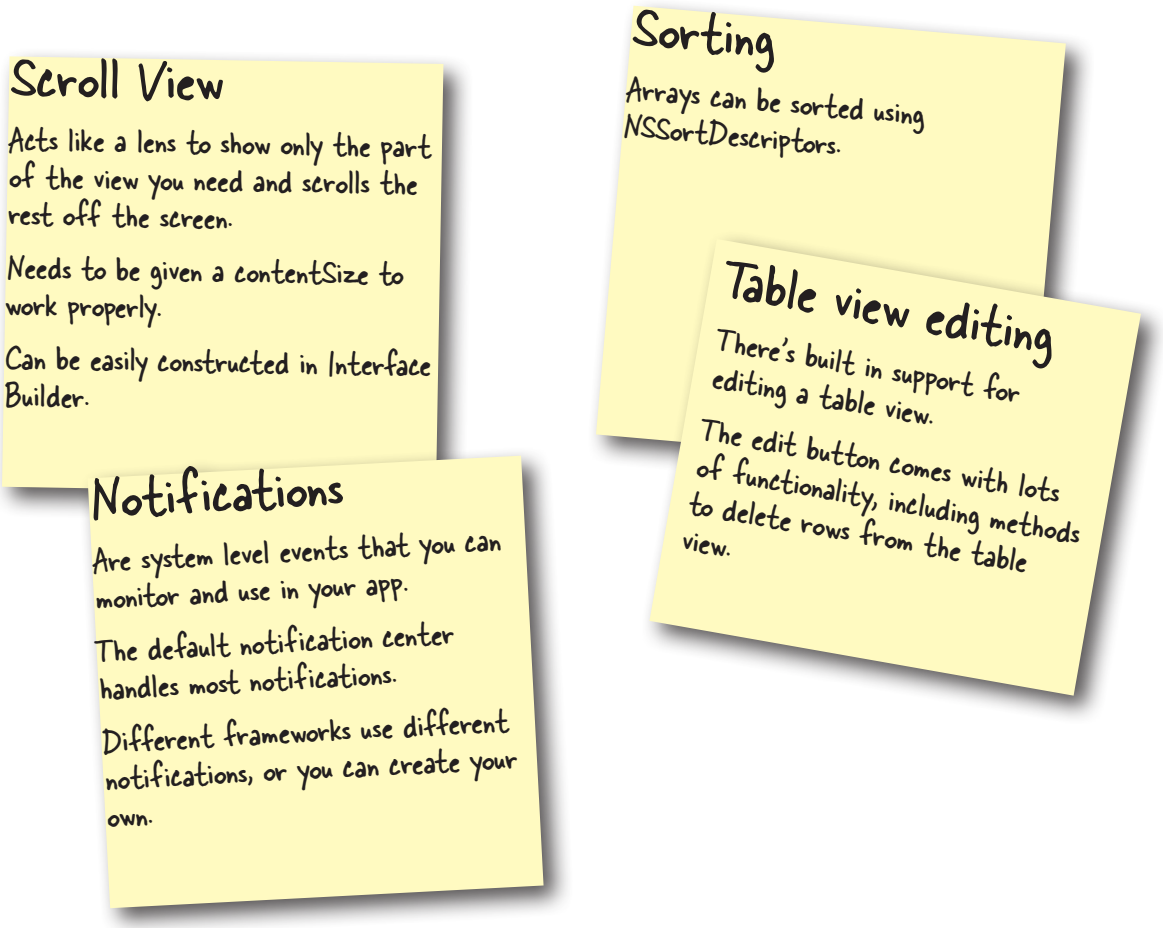
Are pushed onto the stack via the table view or buttons..

Can be subclassed and extended like any other class.

Modal views force the user to interact with them before they can be dismissed.

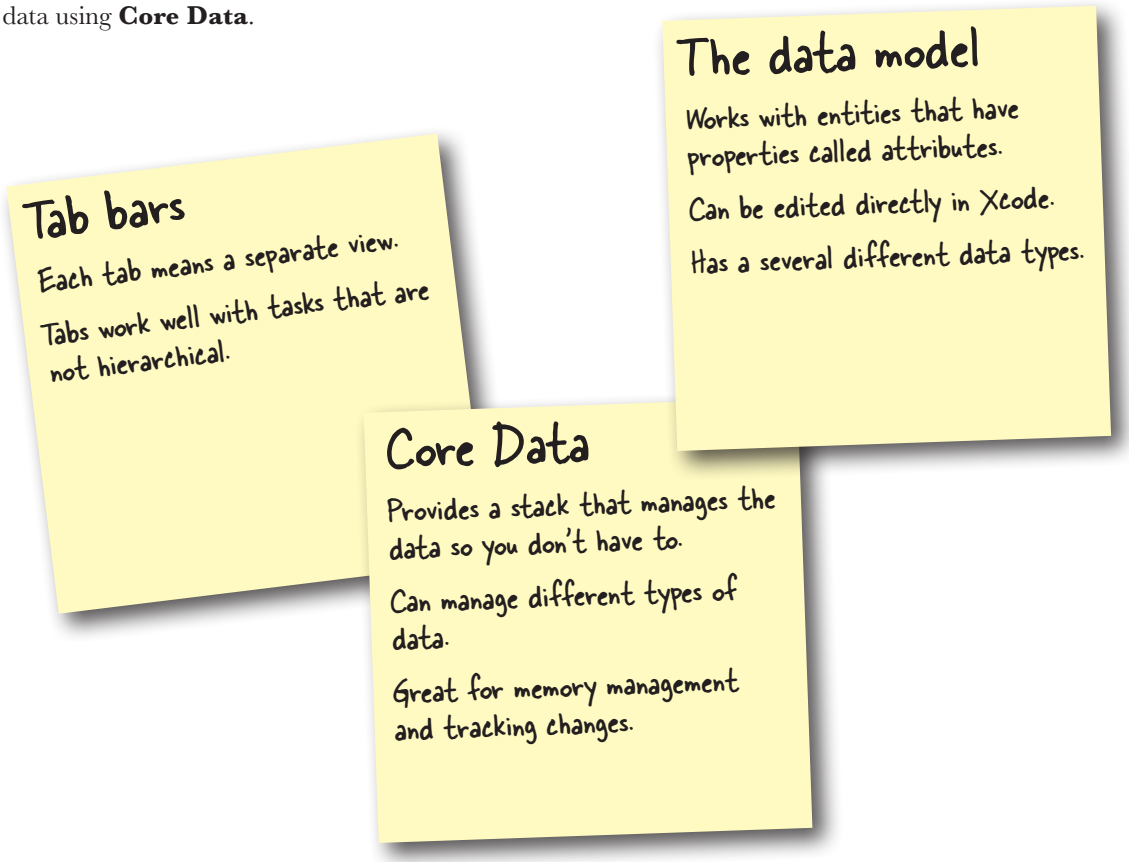
Chapter 6: Editing patterns on iPhone

In this chapter, you got your app really chugging by including a **scroll view**, and supporting **saving**, **editing**, and **sorting** of your app's **data**.



Chapter 7: Wrangling data

In this chapter, you added a **tab bar controller** to your app vocabulary, and also discovered how easy it is to work with data using **Core Data**.



BULLET POINTS

- Core Data is a **persistence framework** that offers loading, saving, versioning and undo-redo.
- **Core Data** can be built on top of SQLite databases, binary files, or temporary memory.
- The **Managed Object Model** defines the **Entities** we're going to ask Core Data to work with.
- The **Managed Object Context** is our entry point to our data. It keeps track of active **Managed Objects**.
- The Managed Object Context is part of the Core Data stack that handles reading and writing our data.

Chapter 8: Moving on up...

In this chapter, you built on the Core Data work from the previous chapter by **migrating** and **versioning** your data. You also used some **new classes** to make **fetching** and **sorting** your data super easy and efficient.

Chapter 9: Justice prevails!

In this chapter, you completed a really big, complex app by adding **Core Location**, the **camera** and **image picker**, and **MapKit** to iBountyHunter.

Flip Animation.

Comes with UIKit.

Is the typical interface for utility apps on iPhone.

Is usually implemented as a modal view.

Camera

Is accessed through the UIImagePickerController.

Is not on all devices, like the iPod Touch, and you need to handle that.

Allows you to select and edit an image for use in your app directly from your library.