Objective-C Fundamental

Christopher K. Fairbairn
Johannes Fahrenkrug
Collin Ruffenach

Novatec

Editor: Rubens Prates
Tradução: Rafael Zanolli
Revisão técnica: Edgard Damiani
Revisão gramatical: Marta Almeida de Sá
Editoração eletrônica: Carolina Kuwabata

ISBN: 978-85-7522-291-1

Histórico de impressões:

Março/2012 Primeira edição

Novatec Editora Ltda.
Rua Luís Antônio dos Santos 110
02460-000 – São Paulo, SP – Brasil
Tel.: +55 11 2959-6529
Fax: +55 11 2950-8869
E-mail: novatec@novatec.com.br
Site: www.novatec.com.br
Twitter: twitter.com/novateceditora
Facebook: facebook.com/novatec
LinkedIn: linkedin.com/in/novatec

Dados Internacionais de Catalogação na Publicação (CIP)
(Câmara Brasileira do Livro, SP, Brasil)

Fairbairn, Christopher K.
Objective-C fundamental / Christopher K. Fairbairn, Johannes Fahrenkrug, Collin Ruffenach ; [tradução Rafael Zanolli]. -- São Paulo : Novatec Editora ; London, NY : Manning Publications, 2012.

Título original: Objective-C fundamentals
ISBN 978-85-7522-291-1 (Novatec)

1. iPhone (Smartphone) - Programação 2. Objective-C (Linguagem de programa para computadores) I. Fahrenkrug, Johannes. II. Ruffenach, Collin. III. Título.

12-01472 CDD-005.117

Índices para catálogo sistemático:

1. Objective-C : Linguagem de programas para computadores : Processamento de dados 005.117

VC20120316

CAPÍTULO 1

Criação de seu primeiro aplicativo iOS

Este capítulo aborda

- A compreensão do ambiente de desenvolvimento iOS
- O uso do Xcode e do Interface Builder
- A construção de seu primeiro aplicativo

Como um desenvolvedor que está dando seus primeiros passos na plataforma iOS, você terá de aprender muitas novas tecnologias e novos conceitos em um curto intervalo de tempo. No pelotão de frente dessa sobrecarga de informações encontra-se um conjunto de ferramentas de desenvolvimento com as quais você talvez não esteja familiarizado e uma linguagem de programação moldada por um grupo de empresas e eventos históricos únicos.

Aplicativos iOS costumam ser desenvolvidos em uma linguagem de programação chamada Objective-C e recebem suporte de uma biblioteca chamada Cocoa Touch. Caso você já tenha desenvolvido aplicativos Mac OS X, deve estar habituado aos "primos" dessas tecnologias no ambiente desktop. Ainda assim, é importante notar que as versões iOS dessas ferramentas não fornecem exatamente as mesmas capacidades das versões desktop, sendo importante que você conheça as restrições, limitações e melhorias oferecidas para dispositivos móveis. Em alguns casos, você poderá até precisar desaprender algumas das práticas habituais que costuma utilizar no ambiente desktop.

Para desenvolver aplicativos iOS, grande parte de seu trabalho será feito em um aplicativo chamado *Xcode*. O Xcode 4, versão mais recente do IDE, traz o Interface Builder (para criação da interface do usuário) integrado diretamente nele. O Xcode 4 permite que você crie, gerencie, implemente e depure seus aplicativos durante todo o ciclo de vida de desenvolvimento do software. Quando você estiver criando um aplicativo que ofereça suporte a mais de um tipo de dispositivo iOS, poderá ser necessário apresentar interfaces de usuário levemente diferentes para tipos específicos de dispositivos, ainda que a mesma lógica subjacente seja utilizada em todas as

variantes. Ficará mais fácil obter esse resultado se você estiver utilizando o conceito de separação modelo-visão-controlador, algo em que o Xcode 4 pode ajudá-lo.

Este capítulo aborda os passos necessários para que você possa utilizar essas ferramentas e criar um pequeno jogo para iPhone; mesmo assim, antes de mergulhar diretamente no processo técnico, vamos discutir o histórico das ferramentas de desenvolvimento iOS e algumas das formas pelas quais o desenvolvimento móvel se diferencia do desenvolvimento de aplicativos desktop e web.

1.1 Apresentação das ferramentas de desenvolvimento iOS

Objective-C é um superconjunto estrito da linguagem C, uma linguagem de programação baseada em procedimentos. Isso significa que qualquer programa válido em C será também válido em Objective-C (ainda que não utilize as melhorias oferecidas por essa linguagem).

Objective-C estende a linguagem C oferecendo recursos orientados a objetos. O modelo de programação orientada a objetos tem por base o envio de mensagens a objetos, diferente do modelo utilizado em C++ e Java, que chama métodos diretamente em um objeto. Essa diferença – sutil – é também uma das características vitais que possibilitam muitos dos recursos presentes em Objective-C e que são mais habitualmente encontrados em linguagens dinâmicas como Ruby e Python.

Uma linguagem de programação, entretanto, depende em grande parte dos recursos oferecidos por suas bibliotecas de suporte. Objective-C fornece uma sintaxe que permite a implementação de lógica condicional e criação de loops, mas não oferece suporte inerente à interação com o usuário, ao acesso de recursos em rede ou à leitura de arquivos. Para facilitar esse tipo de funcionalidade sem que tenhamos de implementá-la do zero em cada aplicativo, a Apple incluiu no SDK um conjunto de bibliotecas de suporte conhecidas coletivamente como *Cocoa Touch*. Se você for um desenvolvedor Java ou .NET, pode imaginar a biblioteca Cocoa Touch como semelhante à Java Class Library ou às Base Class Libraries (BCL) em .NET.

1.1.1 Adaptação dos frameworks Cocoa a dispositivos móveis

A Cocoa Touch é formada por diversos frameworks (habitualmente chamados de *kits*). Um framework é uma coleção de classes agrupadas com um propósito ou uma tarefa comum. Os dois principais frameworks que utilizamos em aplicativos iPhone são o Foundation Kit e o UIKit. O Foundation Kit é uma coleção de classes não-gráficas de sistema que consiste em estruturas de dados, recursos de rede, entrada e saída de arquivos, data, hora e funções de tratamento de strings. O UIKit é um framework projetado para auxiliar no desenvolvimento de GUIs com animações elaboradas.

A Cocoa Touch tem por base os frameworks Cocoa existentes, utilizados no desenvolvimento de aplicativos desktop para Mac OS X. Porém, em vez de fazer da Cocoa Touch uma conversão literal para dispositivos móveis, a Apple otimizou os frameworks para aplicativos iPhone e iPod Touch. Nos casos em que considerou que melhorias de funcionalidade, desempenho ou experiência de usuário poderiam ser obtidas, a Apple substituiu por inteiro alguns frameworks Cocoa. O UIKit, por exemplo, substituiu o framework desktop AppKit.

O ambiente de tempo de execução de software para aplicativos iOS nativos pode ser visto na figura 1.1. Se substituirmos iOS por Mac OS X no campo da base e alguns dos frameworks na camada da Cocoa, veremos que se trata essencialmente da mesma pilha de software que encontramos em aplicativos desktop.

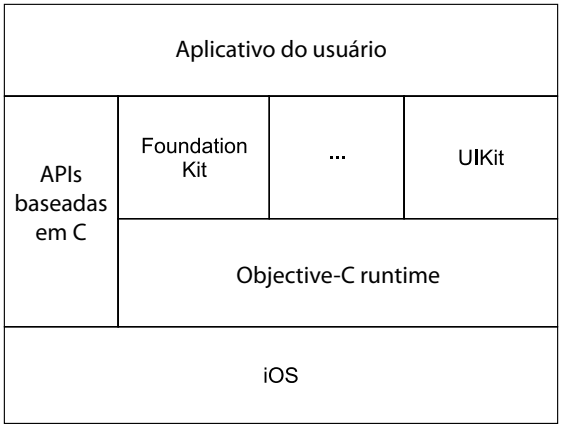

Figura 1.1 – Ambiente em tempo de execução de software para aplicativos iOS, mostrando o sistema operacional, o tempo de execução da linguagem Objective-C e as camadas do framework Cocoa Touch.

Ainda que os frameworks Cocoa Touch sejam APIs baseadas em Objective-C, a plataforma de desenvolvimento iOS também permite que você acesse APIs básicas baseadas em C. A capacidade de reutilizar bibliotecas C (ou C++) em seus aplicativos Objective-C é muito valiosa, pois permite que você reutilize códigos que talvez tenham sido desenvolvidos para outras plataformas móveis, além de possibilitar o uso de várias bibliotecas de código aberto poderosas (desde que suas licenças permitam), o que significa que, em muitos casos, você não terá de "reinventar a roda". Como exemplo, uma rápida pesquisa no Google pode nos mostrar vários exemplos de códigos-fonte em C que implementam recursos de realidade aumentada, análise de imagem e detecção de códigos de barra, apenas para citar algumas possibilidades, sendo que todas podem ser utilizadas diretamente por seu aplicativo Objective-C.

1.2 Ajuste suas expectativas

Por se tratar de um ambiente de desenvolvimento familiar a desenvolvedores Mac OS X, você pode pensar erroneamente que o iPhone é apenas mais um dispositivo de computação portátil, assim como um velho laptop, tablet ou netbook. Isso não poderia estar mais distante da realidade. Um iPhone tem capacidade maior do que um simples celular, mas menor do que um PC desktop padrão. Como dispositivo computacional, ele se encaixa em um nicho de mercado semelhante àquele em que encontramos os netbooks, sendo mais indicado para casos de uso diário, em situações e ambientes diferentes, do que para períodos prolongados de utilização em sessões únicas.

1.2.1 Uma pesquisa de hardware feita em meados de 2011

Em uma primeira análise do iPhone 4, você certamente notará a tela de 3,5 polegadas e 960 x 640 pixels que domina praticamente por inteiro a face frontal do dispositivo. O tamanho geral da tela e o fato de que seus recursos de toque são a única forma de interação com o dispositivo fazem essa característica ter ramificações importantes no design de seus aplicativos. Ainda que a resolução de 960 x 640 pixels seja maior do que a que temos em muitos celulares, provavelmente não será interessante visualizar, nessa tela, uma planilha de 300 colunas por 900 linhas.

Como exemplo do tipo de especificação de hardware que você deve esperar, a tabela 1.1 destaca as especificações dos modelos mais comuns de iPhone, iPod Touch e iPad disponíveis em meados de 2010. No geral, as especificações de hardware ficam alguns anos aquém daquelas dos PCs desktop, mas o número de acessórios de hardware integrados que seus aplicativos podem utilizar (como câmeras, Bluetooth e GPS) é substancialmente maior.

Tabela 1.1 – Comparação das especificações do hardware de diversos dispositivos iPhone e iPod Touch

Recurso	iPhone 3G	iPhone 3GS	iPhone 4	iPad	iPad2
RAM	128 MB	256 MB	512 MB	256 MB	512 MB
Flash	8–16 GB	16–32 GB	16–32 GB	16–64 GB	16–64 GB
Processador	412 MHz ARM11	600 MHz ARM Córtex	1 GHz Apple A4	1 GHz Apple A4	1 GHz dual-core Apple A5
Celular	3,6 Mbps	7,2 Mbps	7,2 Mbps	7,2 Mbps (opcional)	7,2 Mbps (opcional)
Wi-Fi	Sim	Sim	Sim	Sim	Sim
Câmera	2 MP	3 MP AF	5 MP AF (traseira) 0.3 MP (frontal)	–	0.92 MP (traseira) 0.3 MP (frontal)
Bluetooth	Sim	Sim	–	Sim	Sim
GPS	Sim (sem bússola)	Sim	–	Sim (modelos 3G apenas)	Sim (modelos 3G apenas)

Ainda que seja interessante conhecer as capacidades de hardware e as especificações de cada dispositivo, desenvolvedores geralmente não devem se preocupar exageradamente com esses detalhes. À medida que a plataforma iOS amadurece e evolui, novos modelos são lançados, e pode ser difícil se manter atualizado em relação a todas as variações possíveis.

Em vez disso, procure criar um aplicativo capaz de se adaptar, em tempo de execução, ao dispositivo específico em que ele está sendo executado. Sempre que você tiver de utilizar um recurso presente apenas em um subgrupo de dispositivos, teste explicitamente a presença desse recurso e prepare alternativas programáticas para quando ele não estiver disponível. Por exemplo, para determinar se há uma câmera presente, em vez de verificar se o seu aplicativo está sendo executado em um iPhone, é melhor verificar diretamente se a câmera está disponível, uma vez que, agora, alguns novos modelos de iPad já vêm com esse recurso.

1.2.2 Prepare seu aplicativo para quando uma conexão de rede não estiver disponível

Na era de computação em nuvem em que vivemos, muitos aplicativos iOS têm de estar sempre conectados à Internet. A plataforma iOS oferece duas formas principais de conectividade wireless: por área local, com Wi-Fi 802.11, e por área de cobertura, empregando diversos padrões de dados utilizados em celulares. As escolhas de conexão variam muito em velocidade, indo de 300 kilobits a 54 megabits por segundo. Também é possível que a conexão desapareça por completo, como quando o usuário põe o dispositivo em modo de voo, desabilita o roaming durante viagens ou entra em um elevador ou em um túnel.

Diferentemente do ambiente desktop, no qual a maioria dos desenvolvedores simplesmente presume a disponibilidade de uma conexão de rede, bons aplicativos iOS devem ser projetados para que possam se adaptar à falta da rede por longos intervalos de tempo ou mesmo a situações em que a conexão é desfeita de modo inesperado. A pior experiência de usuário para seus clientes será receber uma mensagem de erro do tipo "não foi possível se conectar ao servidor" quando eles estiverem atrasados para uma reunião ou quiserem acessar informações importantes que não exigem obrigatoriamente uma conexão de rede.

Em geral, é importante que você esteja sempre ciente do ambiente em que seu aplicativo iOS está sendo executado. Suas técnicas de desenvolvimento devem considerar não apenas as limitações de memória e processamento do dispositivo, mas também a forma como o usuário interage com seu aplicativo.

Com isso, podemos dizer que já vimos informações básicas suficientes. Agora, vamos mergulhar direto na criação de um aplicativo iOS!

1.3 Uso do Xcode para desenvolver um jogo simples de cara ou coroa

Ainda que você possa ter ideias grandiosas para o próximo sucesso da App Store, vamos iniciar nosso desenvolvimento com um aplicativo relativamente simples, que poderá ser acompanhado com facilidade, sem que você fique preso a muitos detalhes técnicos, e que nos permita ter contato com os recursos exclusivos das ferramentas de desenvolvimento que utilizaremos. No decorrer deste livro, analisaremos mais detalhadamente os pontos específicos que veremos a seguir. Por enquanto, nossa ênfase estará no entendimento do processo geral e não nas especificidades de cada técnica.

O aplicativo que desenvolveremos será um jogo simples que simula um sorteio de cara ou coroa, como aqueles que costumamos ver quando as pessoas têm de tomar uma decisão ou no sorteio do início de uma disputa esportiva. A interface de usuário desse projeto pode ser vista na figura 1.2. Ela é formada por dois botões, *Heads* (cara) e *Tails* (coroa). Utilizando esses botões, o usuário pode iniciar um novo sorteio e escolher sua opção. O iPhone simulará o sorteio e atualizará a tela para indicar se o usuário acertou na escolha. Nosso jogo será chamado "Coin Toss" (Cara ou Coroa).

Figura 1.2 – Coin Toss, o jogo que utilizaremos como exemplo.

No desenvolvimento desse jogo, a primeira ferramenta que devemos estudar é o Xcode.

1.3.1 Apresentação do Xcode – o IDE da Apple

Como mencionamos antes, o Xcode é um IDE que oferece um extenso conjunto de recursos para que você gerencie todo o ciclo de vida de seu projeto de desenvolvimento de software. Criar seu projeto inicial, definir sua classe ou seu modelo de dados, editar o código-fonte, compilar seu aplicativo e, finalmente, depurar e ajustar seu desempenho são todas tarefas que o Xcode pode realizar.

O Xcode se baseia em várias ferramentas de código-aberto, como LLVM (Low-Level Virtual Machine, de código-aberto), GCC (compilador GNU), GDB (depurador GNU) e DTrace (ferramenta de instrumentação e criação de perfis, da Sun Microsystems).

1.3.2 Como iniciar o Xcode com facilidade

Assim que você tiver instalado o kit de desenvolvimento de software (*software development kit*, SDK) do iOS, o primeiro desafio ao utilizar o XCode será localizá-lo. Diferentemente da maioria dos aplicativos que são instalados na pasta */Applications*, a Apple separa ferramentas de desenvolvedores na pasta */Developer/Applications*.

A forma mais fácil de encontrar o Xcode é utilizar o Finder para abrir a pasta raiz Macintosh HD (Figura 1.3). A partir desse ponto, você pode avançar até a pasta *Developer* e, finalmente, à sua subpasta *Applications*. Como desenvolvedor, você passará muito tempo no Xcode, por isso pode ser interessante colocar seu ícone no Dock ou sua pasta na barra lateral do Finder, facilitando seu acesso.

Figura 1.3 – Janela do Finder mostrando a localização da pasta Developer, que contém todas as ferramentas e documentação relacionadas a recursos para desenvolvedores.

Assim que tiver localizado a pasta */Developer/Applications*, você não deverá ter dificuldades em encontrar e iniciar o Xcode.

É importante notar que o Xcode não é sua única opção. Ainda que ele forneça todos os elementos necessários para que você desenvolva seus aplicativos sem necessitar de outros recursos, isso não significa que você não possa complementá-lo com suas ferramentas favoritas. Por exemplo, caso tenha um editor de texto preferido, no qual você se considera mais produtivo, é possível configurar o Xcode para utilizá-lo, em vez da funcionalidade integrada. Se você realmente gosta de sofrer, pode até voltar a utilizar makefiles e a linha de comando.

Socorro! Não encontro o aplicativo Xcode

Se você não encontra uma pasta */Developer*, ou se não vê nenhuma referência aos templates de projetos para iPhone ou iPad quando o Xcode é inicializado, consulte o apêndice A e veja como fazer o download e a instalação do software necessário.

1.3.3 Criação de seu projeto

Para criar seu primeiro projeto, selecione a opção **New Project** no menu **File** (**Shift-Cmd-N**). O Xcode abrirá uma caixa de diálogo New Project, semelhante à da figura 1.4.

Figura 1.4 – Caixa de diálogo New Project do Xcode, mostrando o template View-based Application.

Sua primeira decisão deve ser escolher o tipo de projeto que deseja criar. Isso poderá ser feito selecionando um template que determine o tipo de código-fonte e de configuração que o Xcode deverá adicionar automaticamente ao seu projeto.

Para nosso jogo Coin Toss, você deve escolher o template View-based Application. Para selecioná-lo, escolha primeiro Application sob o título iOS na aba da esquerda e selecione View-based Application. Depois, clique em Next no canto inferior direito. Dê um nome ao seu projeto e especifique o identificador de empresa necessário para associar o aplicativo à sua conta iOS Developer. Para esse projeto, utilize o nome CoinToss e digite um identificador adequado.

O Xcode utiliza os valores referentes ao nome do produto e ao identificador da empresa para criar o que chamamos de *identificador de pacote*. Utilizando essa string, o iOS é capaz de identificar individualmente cada aplicativo. Para que seu sistema operacional permita a execução do CoinToss, seu identificador de pacote deve corresponder a um identificador incluído em um perfil de provisionamento instalado no dispositivo. Se o dispositivo não puder encontrar um perfil adequado, ele se recusará a executar o aplicativo. É dessa forma que a Apple controla, com punho de ferro, quais aplicativos podem ser executados em seus sistemas. Se você não possui um identificador de empresa adequado ou se não está seguro quanto ao que deve digitar nesse campo, siga as instruções do apêndice A antes de prosseguir neste capítulo.

Assim que todos os detalhes tiverem sido digitados, desmarque a caixa Include Unit Tests e clique em Next para selecionar o local em que deseja salvar seu projeto e os arquivos de código-fonte gerados.

Socorro! Não vejo nenhuma opção relacionada ao iOS

Se você não vir nenhum template baseado em iOS no diálogo do novo projeto, é possível que não tenha instalado corretamente o SDK do iOS. Provavelmente, a cópia do Xcode que você está executando é de um DVD de instalação do Mac OS X, ou talvez você tenha feito seu download diretamente no site Apple Developer Connection (ADC). Essa versão é adequada apenas para o desenvolvimento de aplicativos desktop. Instale o SDK iOS da forma descrita no apêndice A e substitua sua cópia do Xcode por uma versão mais adequada que inclua suporte a desenvolvimento para iPhone e iPad.

Você pode estar se perguntando que outros tipos de projeto são possíveis. A tabela 1.2 lista os templates de projetos iOS mais comuns. Sua escolha dependerá do tipo de interface que deseja para seu aplicativo. Não fique muito preocupado com essa seleção; sua decisão não é tão crítica quanto parece. Assim que seu projeto tiver sido criado, você poderá alterar o estilo de seu aplicativo – ainda que isso seja um pouco mais difícil, pois o template do projeto não inserirá automaticamente todo o código-fonte. Nesse caso, você mesmo terá de fazê-lo.

Tabela 1.2 – Templates de projetos disponíveis no Xcode para criação de um novo projeto iOS

Tipo de projeto	Descrição
Navigation-based Application	Cria um aplicativo de estilo semelhante ao Contacts, com uma barra de navegação no topo.
OpenGL ES Application	Cria um aplicativo gráfico baseado em Open GL ES, indicado para jogos e outras funcionalidades.
Split View–based Application	Cria um aplicativo de estilo semelhante ao Mail do iPad. Esse tipo de projeto é indicado para representar informações de estilo master/detail em uma única tela.
Tab Bar Application	Cria um aplicativo de estilo semelhante ao Clock, com uma barra de guias na base.
Utility Application	Cria um aplicativo de estilo semelhante aos aplicativos Stock e Weather, que pode ser virado para revelar uma segunda face.
View-based Application	Cria um aplicativo que consiste em uma única visão. Com essa opção, você pode desenhar e responder a eventos de toque vindos da visão personalizada.
Window-based Application	Cria um aplicativo que consiste de uma única janela, na qual você pode arrastar e soltar controles.

Agora que você completou o diálogo New Project, verá uma janela de projeto semelhante à da figura 1.5. Essa é a janela principal do Xcode. À esquerda, temos um painel Project Navigator e, à direita, um grande painel contextual para edição.

Figura 1.5 – Janela principal do Xcode, com o grupo CoinToss totalmente expandido para revelar os diversos arquivos de código-fonte do projeto.

O painel da esquerda lista todos os arquivos que compõem seu aplicativo. O grupo CoinToss representa o jogo inteiro. Se você expandir esse nó, poderá visualizar subgrupos menores até que eventualmente chegue aos arquivos que formam o projeto. Você está livre para criar seus próprios agrupamentos e organizar os arquivos da forma que preferir.

Quando você clicar em um arquivo no painel da esquerda, o painel da direita será atualizado para fornecer um editor adequado ao arquivo selecionado. Para arquivos de código-fonte, com extensão **.h* e **.m*, um editor de texto tradicional para código-fonte será apresentado. Outros tipos de arquivos (como arquivos **.xib* de recursos) estão associados a editores gráficos mais complexos.

Alguns grupos do painel da esquerda estão associados a comportamentos especiais ou simplesmente não representam arquivos. Por exemplo, os itens do grupo **Frameworks** indicam bibliotecas de código pré-compilado utilizadas pelo projeto atual.

Quando você estiver mais à vontade com o desenvolvimento de aplicativos no Xcode, poderá explorar as muitas seções apresentadas no painel **Project Navigator**. Para iniciar sua jornada, vamos escrever o código-fonte de sua primeira classe.

1.3.4 Elaboração do código-fonte

O template View-based Application apresenta código-fonte suficiente para a criação de um jogo básico para iPhone – tão básico, na verdade, que se você executar o jogo agora mesmo verá simplesmente um retângulo cinza na tela.

Vamos começar a implementação do jogo abrindo o arquivo *CoinTossViewController.h* na janela do Xcode e utilizando o editor de texto para substituir o conteúdo pela listagem que temos a seguir.

Listagem 1.1 – CoinTossViewController.h

```
#import <UIKit/UIKit.h>

@interface CoinTossViewController : UIViewController {
    UILabel *status;
    UILabel *result;
}

@property (nonatomic, retain) IBOutlet UILabel *status;
@property (nonatomic, retain) IBOutlet UILabel *result;

- (IBAction)callHeads;
- (IBAction)callTails;

@end
```

Não se preocupe se o conteúdo dessa listagem não fizer muito sentido para você. No momento, é mais importante que você compreenda o significado global desse código. Os detalhes ficarão por conta do restante do livro – com o tempo, tudo será revelado!

Por ora, vamos nos concentrar em compreender a estrutura geral de um projeto baseado em Objective-C. Objective-C é uma linguagem orientada a objetos, o que significa que grande parte de seu trabalho com o código terá como objetivo definir novas classes (tipos de objetos). A listagem 1.1 define uma nova classe, `CoinTossViewController`. Por convenção, a definição de uma classe é mantida em um arquivo de cabeçalho que utiliza uma extensão **.h*.

No arquivo de cabeçalho `CoinTossViewController`, as duas primeiras linhas declaram que a classe armazena os detalhes de dois controles `UILabel` localizados em algum ponto da interface do usuário. Um `UILabel` pode representar uma única linha de texto, e você utilizará esses rótulos (*labels*) para representar o resultado do cara ou coroa.

O segundo grupo de instruções permite que códigos externos a essa classe digam-lhe quais `UILabels` devem ser utilizados. Finalmente, você especifica que sua classe deve responder a duas mensagens, `callHeads` e `callTails`. Essas mensagens servem para informá-lo se o usuário escolheu cara ou coroa e também para avisá-lo de que um novo sorteio deve ser iniciado.

Um arquivo de cabeçalho (**.h*) especifica aquilo que você deve esperar de uma classe e como outros códigos podem interagir com ela. Agora que você atualizou o arquivo de cabeçalho, deve fornecer a implementação em si dos recursos especificados. Abra o arquivo *CoinTossViewController.m* correspondente e substitua seu conteúdo pelo da listagem a seguir.

Listagem 1.2 – CoinTossViewController.m

```
#import "CoinTossViewController.h"
#import <QuartzCore/QuartzCore.h>

@implementation CoinTossViewController

@synthesize status, result;    ❶ Corresponde à @property

- (void) simulateCoinToss:(BOOL)userCalledHeads {
    BOOL coinLandedOnHeads = (arc4random() % 2) == 0;

    result.text = coinLandedOnHeads ? @"Heads" : @"Tails";

    if (coinLandedOnHeads == userCalledHeads)
        status.text = @"Correct!";
    else
        status.text = @"Wrong!";

    CABasicAnimation *rotation = [CABasicAnimation    ❷ Define dois objetos
        animationWithKeyPath:@"transform.rotation"];
```

```
    rotation.timingFunction = [CAMediaTimingFunction
        functionWithName:kCAMediaTimingFunctionEaseInEaseOut];
    rotation.fromValue = [NSNumber numberWithFloat:0.0f];
    rotation.toValue = [NSNumber numberWithFloat:720 * M_PI / 180.0f];
    rotation.duration = 2.0f;
    [status.layer addAnimation:rotation forKey:@"rotate"];

    CABasicAnimation *fade = [CABasicAnimation    ❸ Afeta o rótulo
        animationWithKeyPath:@"opacity"];
    fade.timingFunction = [CAMediaTimingFunction
        functionWithName:kCAMediaTimingFunctionEaseInEaseOut];
    fade.fromValue = [NSNumber numberWithFloat:0.0f];
    fade.toValue = [NSNumber numberWithFloat:1.0f];
    fade.duration = 3.5f;
    [status.layer addAnimation:fade forKey:@"fade"];
}

- (IBAction) callHeads {
    [self simulateCoinToss:YES];
}

- (IBAction) callTails {
[self simulateCoinToss:NO];
}

- (void) viewDidUnload {
    self.status = nil;
    self.result = nil;
}

- (void) dealloc {      ❹ Gerenciamento de memória
    [status release];
    [result release];
    [super dealloc];
}

@end
```

À primeira vista, a listagem 1.2 pode lhe parecer longa e assustadora, mas, dividindo-a em passos menores, você verá que não é difícil entender o que ela faz.

A primeira instrução, ❶, corresponde ao código das declarações `@property` em *CoinTossViewController.h*. O conceito de propriedades e as vantagens de propriedades sintetizadas serão explorados mais detalhadamente no capítulo 5.

A maioria da lógica do arquivo *CoinTossViewController.m* está contida no método `simulateCoinToss:`, chamado sempre que o usuário deseja o resultado de um novo sorteio. A primeira linha simula um sorteio gerando um número aleatório, entre `0` e `1`, para representar cara e coroa, respectivamente. O resultado é armazenado em uma variável que chamamos de `coinLandedOnHeads`.

Uma vez determinado o resultado do sorteio, os dois controles `UILabel` da interface do usuário são atualizados. A primeira instrução condicional atualiza o rótulo do resultado para indicar o resultado do sorteio simulado; a segunda indica se o usuário acertou em sua escolha.

O restante do método `simulateCoinToss:` prepara dois objetos `CABasicAnimation` ❷, e ❸, para que o estado do sorteio seja mostrado e depois desapareça gradualmente, em vez de ser atualizado abruptamente. Isso é feito solicitando-se que a propriedade `transform.rotation` do controle `UILabel` gire suavemente de `0` a `720` graus em `2.0` segundos, enquanto a propriedade `opacity` passa de 0% (`0.0`) a 100% (`1.0`) durante `3.5` segundos. É importante que você perceba que essas animações são realizadas de modo declarativo. Você especifica a alteração ou o efeito que deseja e deixa que o framework se preocupe com a lógica de timing e redesenho necessários para implementar esses efeitos.

O método `simulateCoinToss:` espera um único parâmetro, `userCalledHeads`, o qual indica se o usuário deseja que o sorteio resulte em cara ou coroa. Dois métodos adicionais, `callHeads` e `callTails`, são métodos de conveniência que chamam `simulateCoinToss:`, com o parâmetro `userCalledHeads` definido como esperado.

O método final, `dealloc` ❹, lida com questões relacionadas ao gerenciamento de memória. Discutiremos esse tópico detalhadamente no capítulo 9. É importante que você perceba que a linguagem Objective-C não coleta automaticamente a memória não utilizada (ao menos no que se refere ao iPhone). Isso significa que, se você alocar memória ou recursos do sistema, também será responsável por liberá-los (ou desalocá-los). Se não o fizer, seu aplicativo acabará consumindo artificialmente mais recursos do que necessário e, nos piores casos, consumirá por completo os recursos limitados do dispositivo, travando o aplicativo.

Agora que desenvolveu a lógica básica do jogo, você deve criar a interface de usuário no Xcode e conectá-la ao código da classe `CoinTossViewController`.

1.4 Conexão da interface do usuário

Neste estágio você pode perceber, pela definição da classe `CoinTossViewController`, que a interface do usuário deve ter ao menos dois controles `UILabel` e invocar as mensagens `callHeads` ou `callTails` sempre que o usuário quiser o resultado de um novo sorteio. Você ainda não especificou em que ponto da tela os rótulos devem ser posicionados ou como o usuário pode solicitar a realização de um sorteio.

Há dois modos de especificar esse tipo de detalhe. Primeiro, poderíamos escrever o código-fonte que cria os controles da interface do usuário, configurar suas propriedades, como tamanho de fonte e cor, e posicioná-los na tela. Certamente, demoraríamos a

escrever esse código e você poderia gastar muito de seu tempo simplesmente tentando imaginar o visual adequado dessa implementação.

A melhor alternativa é utilizar o Xcode, que permite que você visualize o layout e configure os controles de sua interface de usuário, conectando-os ao código-fonte. A maioria dos templates de projetos iOS utiliza essas técnicas, e geralmente inclui um ou mais arquivos **.xib* projetados para descrever a interface do usuário. Esse projeto não é uma exceção; clique no arquivo *CoinTossViewController.xib* no painel Project Navigator e veja que o painel do editor exibe seu conteúdo (Figura 1.6).

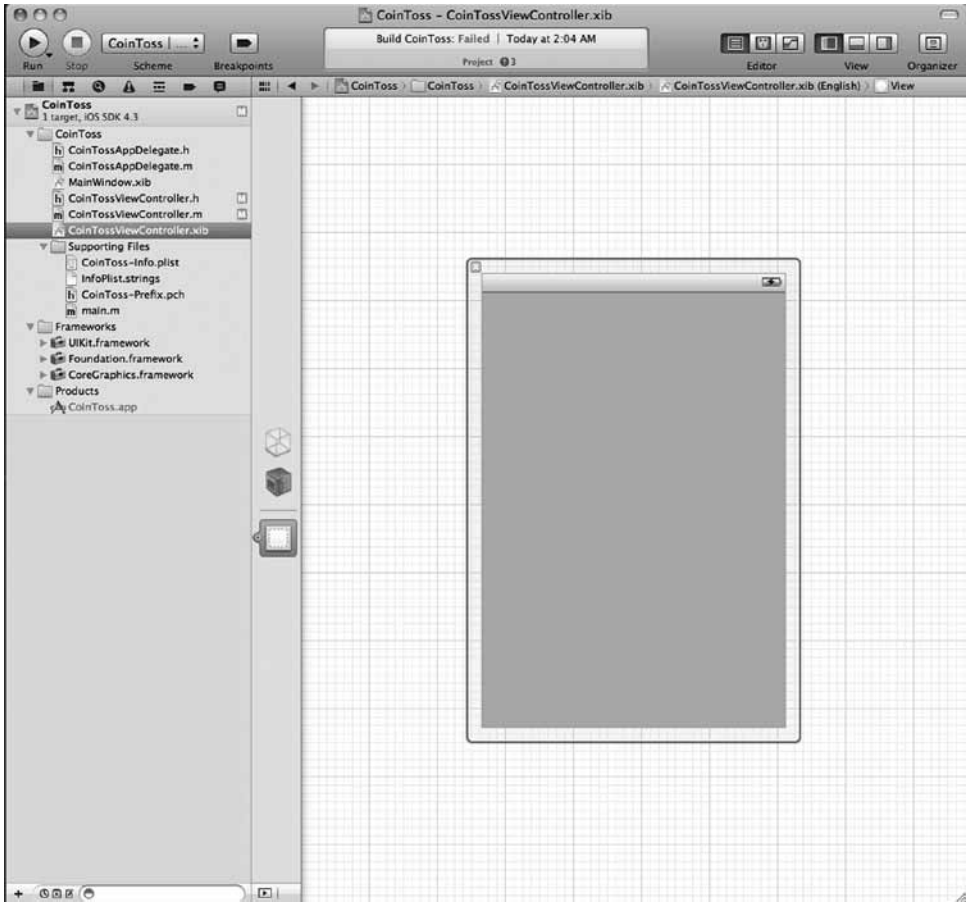

*Figura 1.6 – Janela principal do Xcode mostrando a edição de um arquivo *.xib. Na borda esquerda do editor você pode ver três ícones, cada um representando um objeto ou um componente GUI diferente, armazenado no arquivo .xib.*

Na borda esquerda do painel do editor temos alguns ícones. Cada um representa um objeto criado quando o jogo é executado, e cada um tem uma dica de contexto (*tooltip*) que mostra seu nome. A caixa representada apenas por suas arestas, intitulada File's

Owner, representa uma instância da classe `CoinTossViewController`; o retângulo branco representa a visão (ou a tela) principal do aplicativo. Utilizando o Xcode, você pode configurar graficamente as propriedades desses objetos e criar conexões entre eles.

1.4.1 Inclusão de controles a uma visão

O primeiro passo na definição da interface do usuário de seu jogo será o posicionamento dos controles necessários dentro da visão.

Para incluir controles, localize-os na janela **Library**, a qual contém um catálogo de controles disponíveis para a interface do usuário. Arraste-os e solte-os na visão. Se a janela **Library** não estiver visível, você pode abri-la com a opção de menu **View > Utilities > Object Library** (Control-Option-Cmd-3). Para o jogo de cara ou coroa, você necessita de dois **Labels** e dois **Rounded Rect Buttons**. Arraste dois de cada para a visão. O processo de arrastar e soltar um controle na visão pode ser visto na figura 1.7.

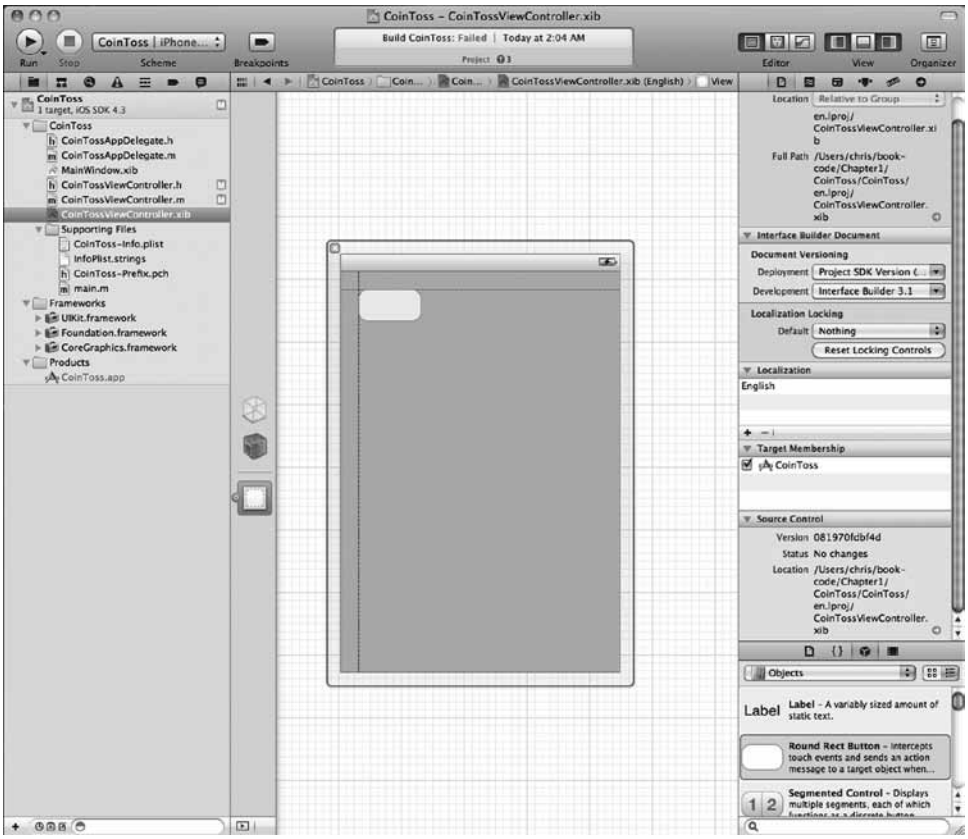

Figura 1.7 – Arrastando e soltando controles na visão. Note as linhas de ajuste que servem para garantir que sua interface de usuário esteja de acordo com as diretrizes para interface humana (Human Interface Guidelines, HIG) do iOS.

Depois de arrastar e soltar os controles na visão, você pode redimensioná-los e ajustar suas posições de acordo com a estética pretendida. A forma mais fácil de alterar o texto mostrado por um botão ou um controle é clicar duas vezes nele e digitar o texto. Para alterar outras propriedades, como o tamanho e a cor das fontes, você pode utilizar o painel Attributes Inspector, que pode ser acessado pela opção de menu View > Utilities > Attributes Inspector (Alt-Cmd-4). Quando estiver estilizando sua visão, consulte a figura 1.2 para referência.

Com os controles posicionados na interface do usuário, a única tarefa que resta é conectá-los ao código que escrevemos. Lembre-se de que a classe definida no arquivo de cabeçalho *CoinTossViewController.h* requer três elementos da interface do usuário:

- um local para enviar as mensagens `callHeads` ou `callTails` sempre que o usuário quiser iniciar um sorteio;
- um `UILabel` para mostrar o resultado do último sorteio (cara ou coroa);
- um `UILabel` para mostrar o estado do último sorteio (correto ou incorreto).

1.4.2 Conexão dos controles ao código-fonte

A interface de usuário que acabamos de criar atende a esses requisitos, mas o código não consegue determinar quais botões devem indicar que o usuário escolheu cara ou coroa (mesmo que o texto dos botões torne isso óbvio para um humano). Nesse caso, você deve estabelecer explicitamente essas conexões. O Xcode permite que você o faça de modo gráfico.

Segure a tecla Control e arraste o botão rotulado como Heads em direção ao ícone que representa a instância de `CoinTossViewController` (File's Owner) localizada na borda esquerda do editor. Conforme você arrasta, uma linha azul deve aparecer entre os dois elementos.

Quando soltar o mouse, um menu pop-up será mostrado, permitindo que você escolha qual mensagem deve ser enviada ao objeto `CoinTossViewController` sempre que o botão for pressionado (Figura 1.8). Nesse caso, você seleciona `callHeads`, uma vez que essa é a mensagem que corresponde à intenção do botão.

Você pode repetir esse processo para conectar o botão Tails ao método `callTails`. Essas duas conexões fazem com que o pressionamento de cada botão na interface do usuário resulte na execução de lógica na classe `CoinTossViewController`. Ter essas conexões especificadas graficamente, em vez de programaticamente, representa uma abordagem muito flexível, pois permite que você experimente, com rapidez e facilidade, conceitos distintos de interface de usuário, alterando seus controles e conectando-os novamente à classe.

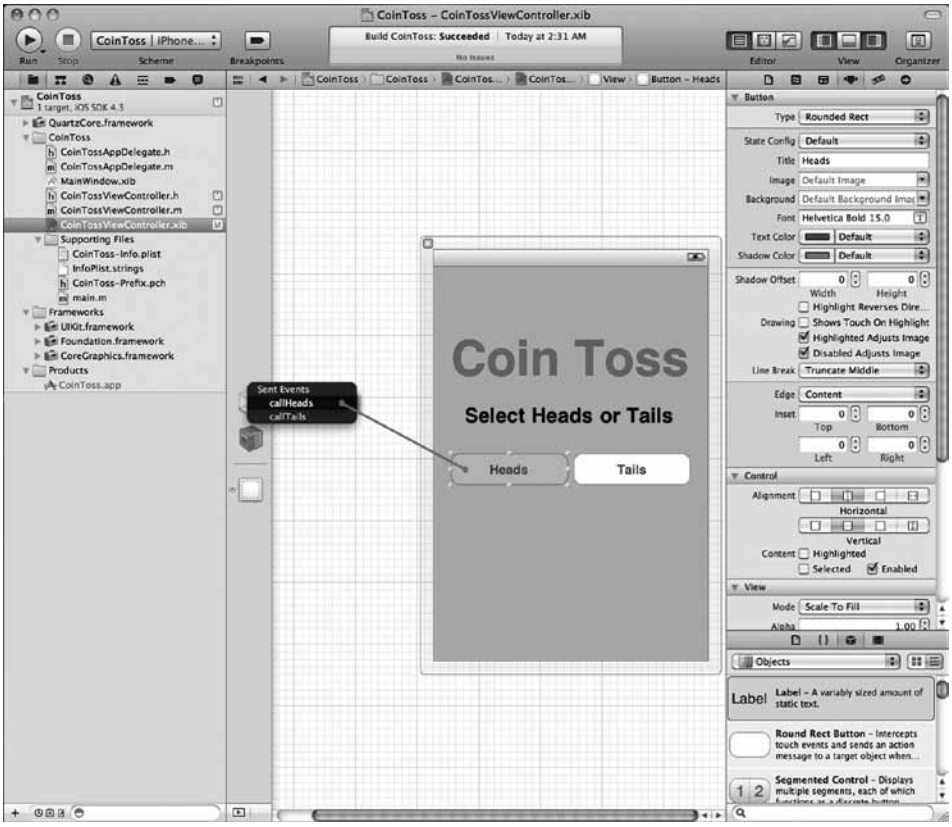

Figura 1.8 – Formando visualmente uma conexão entre o controle do botão e a classe CoinTossViewController, com o recurso de arrastar e soltar.

Se o Xcode se recusar a efetuar uma conexão entre um controle da interface do usuário e um objeto, pode ser que haja um problema no código-fonte: talvez um simples erro de digitação ou um tipo de dado incorreto. Nesse caso, certifique-se de que o aplicativo ainda pode ser compilado e corrija os erros que surgirem antes de efetuar a conexão novamente.

Terminado seu trabalho com os botões, agora você terá de conectar os controles dos rótulos à classe `CoinTossViewController` para permitir que o código atualize a interface do usuário com os resultados do último sorteio.

Para conectar os controles dos rótulos, você também pode utilizar uma operação de arrastar e soltar como a que vimos. Dessa vez, segurando a tecla **Control**, clique no ícone que representa a instância de `CoinTossViewController` arrastando-o até o rótulo na visão. Ao soltar o mouse, surgirá um menu pop-up permitindo a seleção da propriedade da classe `CoinTossViewController` que você deseja conectar ao controle do rótulo. Esse processo pode ser visto na figura 1.9. Utilizando-se dele, conecte o rótulo **CoinToss** à propriedade `status` e o rótulo intitulado **Select Heads or Tails** à propriedade `result`.

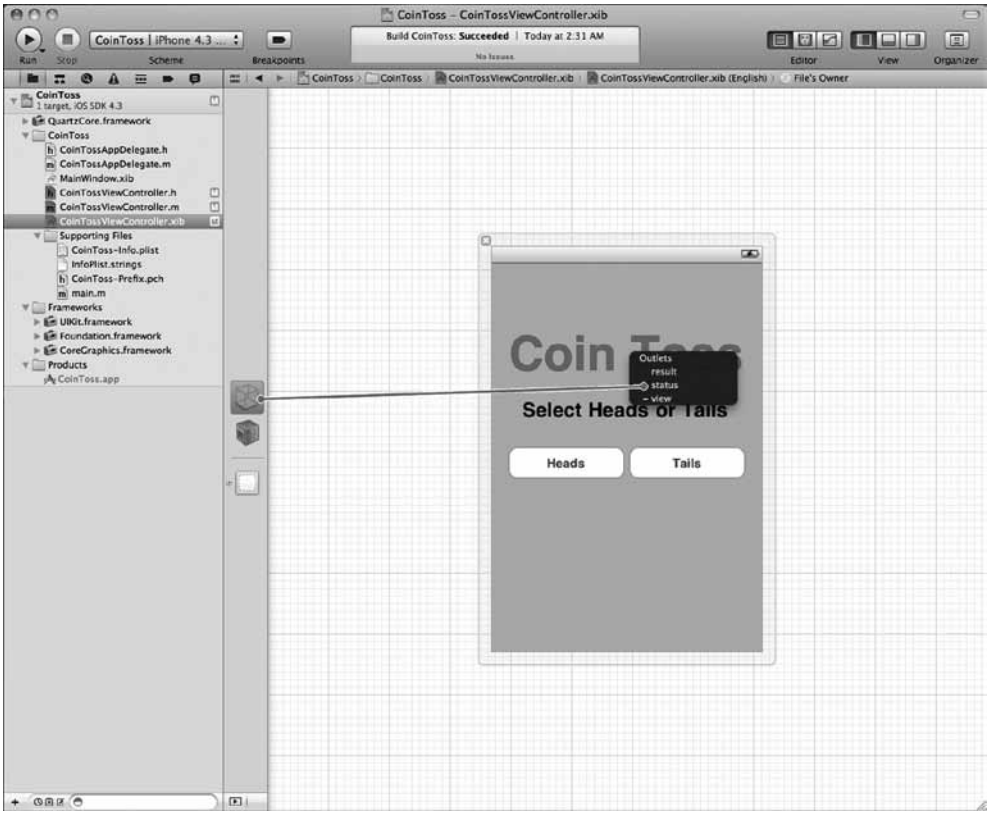

Figura 1.9 – Formando visualmente uma conexão entre a variável de instância de estado e o controle do rótulo na interface do usuário, com o recurso de arrastar e soltar (segurando a tecla Control).

Para decidir como formar as conexões dos objetos, pense no fluxo das informações. No caso de um botão, pressioná-lo fará com que um método do aplicativo seja executado, ao passo que, no caso da conexão do rótulo, uma alteração no valor da variável de instância da classe deverá atualizar a interface do usuário.

Você pode estar se perguntando como o Xcode determina quais itens devem ser mostrados no menu pop-up. Se você retornar à listagem 1.1, a resposta poderá ser vista nas palavras-chave especiais `IBOutlet` e `IBAction`. O Xcode processa o código-fonte e permite que você conecte a interface do usuário a qualquer elemento marcado com um desses atributos especiais.

Nesse estágio, pode ser interessante verificar se você efetuou corretamente as conexões necessárias. Se você segurar `Control` e clicar no ícone que representa a instância de `CoinTossViewController`, verá um menu pop-up que lhe permite verificar como todas as saídas e ações associadas a um objeto estão conectadas. Se você passar o ponteiro do mouse sobre uma das conexões, o Xcode destacará o objeto associado. Esse recurso pode ser visto na figura 1.10.

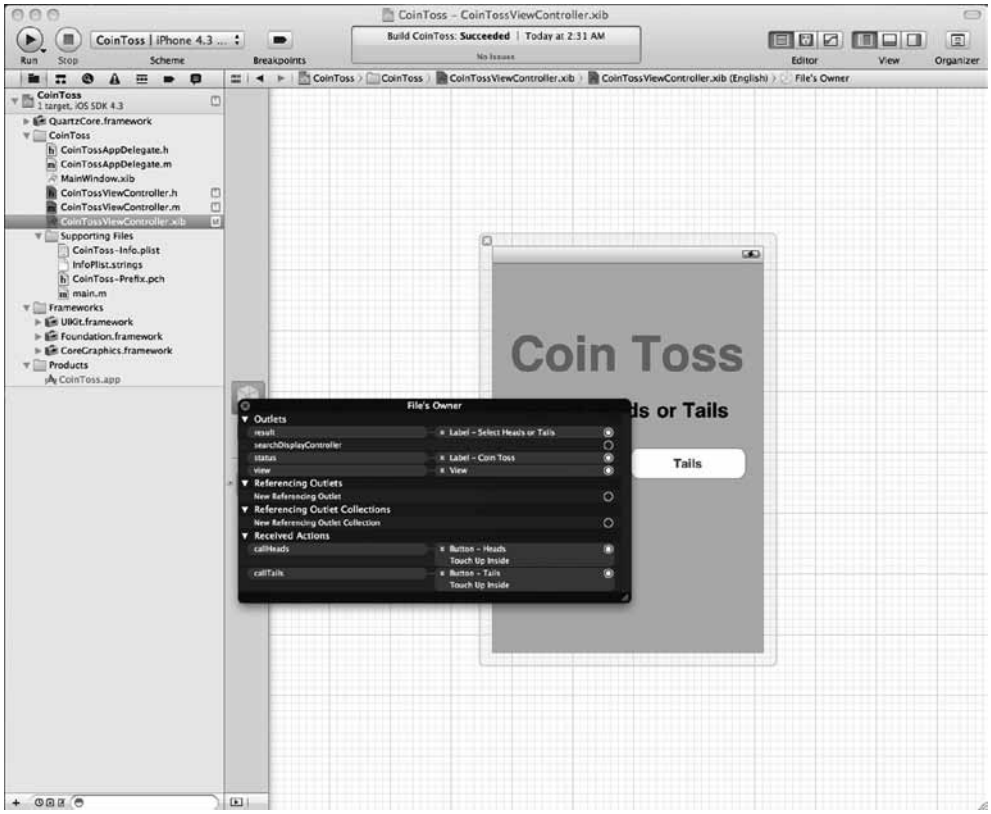

Figura 1.10 – Verificação das conexões feitas de, e para o objeto CoinTossViewController.

Com isso, podemos dizer que você concluiu seu trabalho na interface do usuário. Agora, seguiremos em frente, verificando se cometemos algum erro, e finalmente vendo se nosso jogo funciona corretamente.

NIBs vs. XIBs

A interface de usuário para um aplicativo iOS é armazenada em um arquivo .xib. Ainda assim, na documentação e nos frameworks Cocoa Touch, esses arquivos são geralmente chamados de *nibs*.

Não há problema em trocar esses termos: um arquivo .xib utiliza um formato mais novo de arquivo baseado em XML, o que facilita o armazenamento do arquivo em sistemas de controle de revisão e assim por diante.

Um .nib, por outro lado, representa um formato binário mais antigo, que resulta em arquivos menores, com maior velocidade de processamento, e assim por diante.

A documentação geralmente se refere a arquivos NIB em vez de XIB, uma vez que, à medida que compila seu projeto, o Xcode converte automaticamente arquivos *.xib para o formato *.nib.

1.5 Compilação do jogo de cara ou coroa

Agora que terminou a codificação de seu aplicativo, você deverá converter seu código-fonte para um formato que o iPhone possa utilizar. Esse processo é chamado de *compilação* do projeto. Para compilar o jogo, selecione **Build** no menu **Product** (ou pressione Cmd-B).

Enquanto o projeto é compilado, você pode monitorar o processo verificando o indicador na barra de ferramentas. Nele, você deve encontrar a seguinte mensagem: "Build CoinToss: Succeeded". Se você cometeu algum erro, clique no ícone vermelho de exclamação sob o texto (ou pressione Cmd-4) para visualizar uma lista de erros e avisos que devem ser resolvidos.

Ao clicar em um erro nessa lista, você terá acesso ao código-fonte correspondente, no qual as linhas com problemas estarão destacadas, como na figura 1.11. Depois de corrigir o que há de errado no código, você poderá compilar o aplicativo novamente, repetindo esse processo até que todos os problemas tenham sido solucionados.

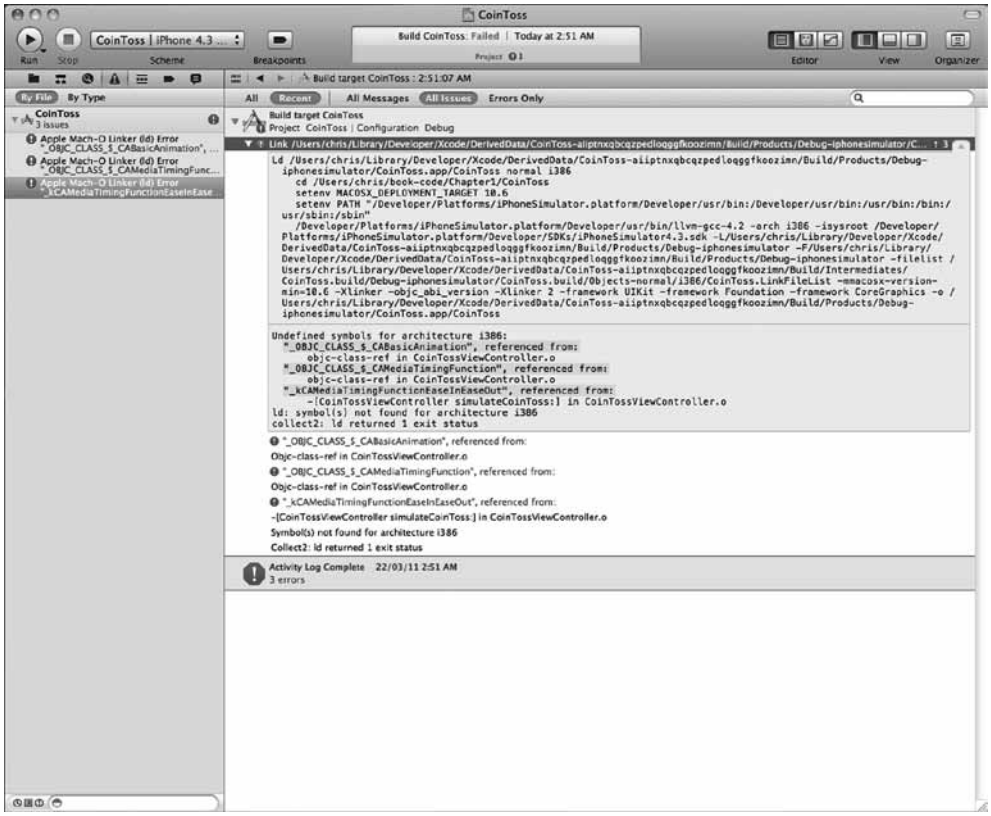

Figura 1.11 – O editor de texto do Xcode destaca visualmente linhas do código-fonte que apresentam erros de compilação. Depois de corrigir os erros, a compilação do projeto indicará se você corrigiu o problema.

Ao compilar o jogo de cara ou coroa, você deve notar a presença de erros que mencionam `kCAMediaTimingFunctionEaseInEaseOut`, `CAMediaTimingFunction` e `CABasicAnimation`. Para corrigir esses erros, selecione o projeto CoinToss no Project Navigator (primeiro item na visão em árvore). No editor que surge para esse item, vá até a aba Build Phases e expanda a seção Link Binary with Libraries. A região expandida deve mostrar uma lista de frameworks adicionais exigidos por seu aplicativo. Para que as animações de interface funcionem, você terá de clicar no botão +, na base da janela, e selecionar `QuartzCore.framework` da lista apresentada.

Para manter tudo organizado, assim que você tiver adicionado a referência ao framework QuartzCore, pode ser interessante movê-lo dentro da visão em árvore do navegador do projeto, de forma a ficar sob a seção Frameworks, ao lado de outros frameworks dos quais seu aplicativo depende.

1.6 Um teste do Coin Toss

Agora que você compilou o jogo e corrigiu seus erros mais óbvios, estamos prontos para verificar se nosso projeto funciona como pretendido. Para isso, você poderia simplesmente executar o jogo e verificar se ele se comporta corretamente ou se trava, mas isso não seria muito prático e, além do mais, acabaria por obrigá-lo a adivinhar o que acontece no código. Para auxiliá-lo nessa situação, o Xcode fornece um depurador integrado que se conecta à execução de seu aplicativo, permitindo que você o pause temporariamente para observar o valor das variáveis, percorrendo o código-fonte linha a linha. Porém, antes de aprender a utilizar esse recurso, faremos um breve desvio.

1.6.1 Seleção de um destino

Antes de testar seu aplicativo, você deve decidir onde deseja executá-lo. Durante as fases iniciais de desenvolvimento, você geralmente testará seu aplicativo por meio do iOS Simulator. Esse simulador faz a função de um dispositivo iPhone ou iPad e pode ser executado em uma janela do desktop de sua máquina Mac OS X. Utilizar o simulador pode acelerar o desenvolvimento de seu aplicativo, uma vez que é muito mais rápido para o Xcode transferir e depurar seu aplicativo no simulador do que trabalhar com um iPhone real.

Desenvolvedores com experiência em outras plataformas móveis podem estar acostumados a utilizar emuladores de dispositivos. Os termos *simulador* e *emulador* não são sinônimos. Diferentemente de um emulador, que procura emular o dispositivo no nível do hardware (e que, dessa forma, é capaz de executar um firmware virtualmente idêntico ao do dispositivo real), um simulador procura apenas fornecer um ambiente que tenha um conjunto compatível de APIs.

Sempre faça seus testes em um dispositivo iPhone, iPod Touch ou iPad real

Os códigos de exemplo deste livro foram projetados para ser executados no iOS Simulator. Essa é uma forma rápida e fácil de desenvolver seu aplicativo de modo iterativo, sem que você tenha de se preocupar com a conectividade do dispositivo ou com a demora de sua transferência para um dispositivo real.
Como o iOS Simulator não é uma réplica perfeita de um iPhone, é possível que um aplicativo funcione no simulador, mas falhe em um dispositivo real. Nunca publique um aplicativo na App Store do iTunes sem testá-lo em um dispositivo real, ou, melhor ainda, procure testar seu aplicativo em alguns modelos diferentes de dispositivos, como o iPhone e o iPod Touch.

O iOS Simulator executa seu aplicativo na cópia do Mac OS X utilizada por seu desktop, o que significa que ocasionalmente podem haver diferenças entre o simulador e um iPhone de verdade. Um exemplo simples de uma situação em que isso ocorre pode ser visto no que se refere a nomes de arquivos. No iOS Simulator, nomes de arquivos geralmente não diferenciam maiúsculas de minúsculas, ao passo que, em um iPhone, essa diferença se verifica.

Por padrão, a maioria dos templates de projetos está configurada para implantar seu aplicativo no iOS Simulator. Para implantar seu aplicativo em um iPhone real, você deve alterar o destino, de iPhone Simulator para iOS Device. A forma mais fácil de fazê-lo é selecionar o destino desejado no menu suspenso que temos na parte esquerda da barra de ferramentas, na janela principal do Xcode (Figura 1.12).

Figura 1.12 – Canto superior esquerdo da janela principal do Xcode. A seleção de CoinToss/ iPhone 4.3 Simulator, oferecida pelo menu suspenso, permite que você alterne entre o iPhone Simulator e o dispositivo iOS.

Ao definir o destino como iOS Device, você garantirá que o Xcode tentará implantar seu aplicativo em seu iPhone real. Ainda assim, será necessário mais uma alteração antes que isso dê certo.

1.6.2 Uso de breakpoints para inspecionar o estado de um aplicativo em execução

Quando você está testando um aplicativo, é comum que queira investigar o comportamento de uma seção específica do código-fonte. Antes de iniciar o aplicativo, pode ser interessante configurar o depurador para pausar automaticamente a execução sempre que pontos como esse forem atingidos. Isso pode ser feito por meio de um recurso que chamamos de *breakpoints*, ou pontos de interrupção.

Para o depurador, um breakpoint indica um ponto no código-fonte em que o usuário gostaria de interromper a execução e verificar o código, explorando o valor atual das variáveis, e assim por diante.

Para nosso jogo de cara ou coroa, vamos adicionar um breakpoint ao início do método `simulateCoinToss:`. Abra o arquivo *CoinTossViewController.m* e vá até a seção que implementa esse método. Se você clicar na margem esquerda ao lado da primeira linha, deverá fazer com que apareça, nesse ponto, um pequeno marcador azul (Figura 1.13).

Figura 1.13 – Definição de um breakpoint para uso pelo depurador sempre que a primeira linha do método `simulateCoinToss:` *for chamada. Note a presença do marcador na margem, indicando um breakpoint ativo.*

O marcador azul indica que essa linha tem um breakpoint habilitado. Se você clicar no breakpoint, sua cor mudará para um azul mais claro, indicando que ele foi desabilitado, o que fará com que o depurador o ignore até que você clique novamente nele, reabilitando-o. Para remover permanentemente um breakpoint, clique sobre ele e arraste-o para fora da margem. Ao soltar o mouse, você verá uma pequena animação indicando o desaparecimento do breakpoint, e ele será removido.

1.6.3 Execução do jogo CoinToss no simulador do iPhone

Com o breakpoint posicionado, você finalmente estará pronto para executar o aplicativo e vê-lo em ação. Selecione Run do menu Product (Cmd-R). Depois de alguns segundos, o aplicativo aparecerá em seu iPhone. Todo o seu trabalho valeu a pena. Parabéns! Você agora é oficialmente um desenvolvedor iPhone!

Se você quiser executar o jogo, mas não quiser que nenhum de seus breakpoints esteja habilitado, pode clicar em cada um para desabilitá-los individualmente. Isso seria um pouco demorado e exigiria que você reabilitasse manualmente todos os breakpoints quando quisesse utilizá-los novamente. Como alternativa prática, você pode desabilitá-los temporariamente selecionando a opção Product > Debug > Deactivate Breakpoints (Cmd-Y).

1.6.4 Controle do depurador

Agora que você viu seu primeiro aplicativo iPhone em execução, estou certo de que não resistiu e apertou um dos botões de cara ou coroa. Ao tocar em um botão, note que a janela do Xcode pula para o primeiro plano. Isso ocorre porque o depurador detecta que a execução do aplicativo atingiu um ponto onde inserimos um breakpoint.

A janela do Xcode que surge deve ser semelhante à da figura 1.14. Note que o painel principal da janela do Xcode mostra o código-fonte do método que está sendo executado. Ao passar seu mouse sobre uma variável do código-fonte, você verá uma dica de dados, mostrando o valor atual da variável. A linha do código-fonte que está prestes a ser executada será destacada e identificada por uma seta verde na margem direita.

Enquanto o depurador estiver sendo executado, você deverá notar uma alteração no painel esquerdo da janela do Xcode para mostrar a pilha de chamadas de cada thread do aplicativo. A pilha de chamadas lista a ordem na qual os métodos executados foram chamados, tendo o método atual no topo. Muitos dos métodos listados estarão em cinza, indicando que o código-fonte não está disponível para eles; nesse caso, isso ocorre porque muitos deles são detalhes internos do framework Cocoa Touch.

Um novo painel na base da tela também deve estar visível; ele mostra os valores atuais das variáveis e argumentos relevantes à posição atual do depurador, assim como saídas em texto emitidas pelo depurador (Figura 1.14).

Na parte superior do painel localizado na base do depurador, você poderá notar uma série de pequenos botões formando uma barra de ferramentas, como a que vemos na figura 1.15.

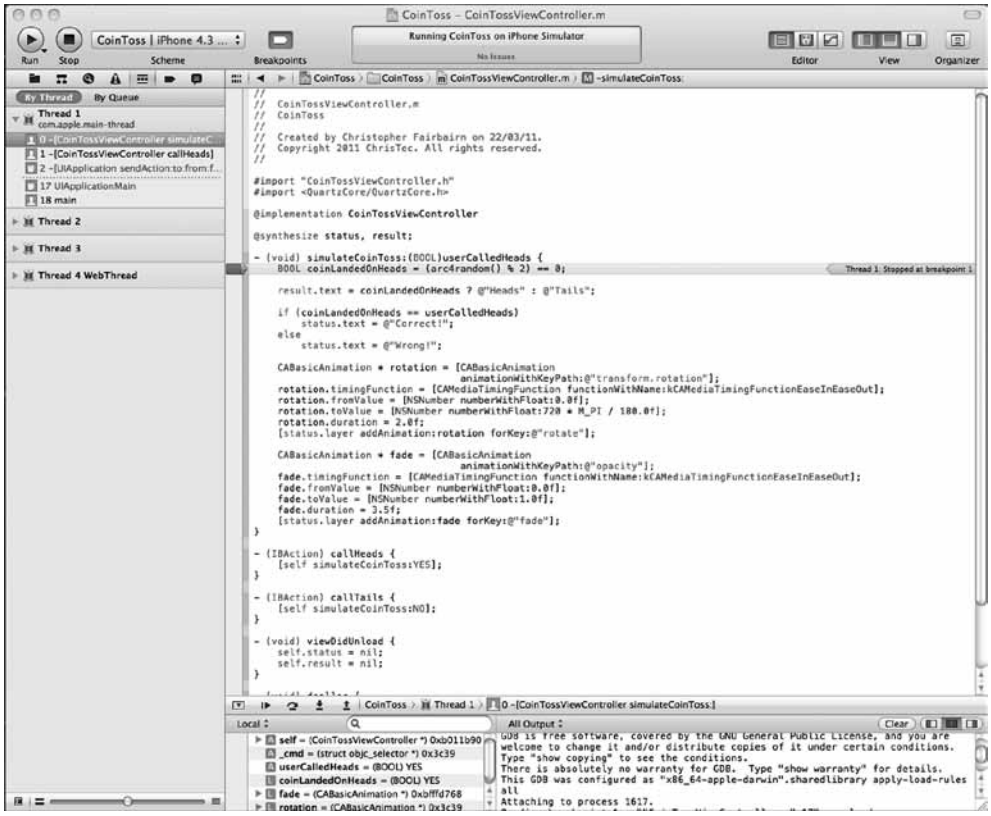

Figura 1.14 – Janela do depurador do Xcode, depois que a execução atingiu um breakpoint.

Figura 1.15 – Opções da barra de ferramentas do Xcode para controlar o depurador.

Essas opções permitem que você controle o depurador e ganham importância quando ele pausa o aplicativo ou interrompe sua execução ao atingir um breakpoint. Esses botões (que nem sempre podem estar todos presentes) permitem que você realize as seguintes ações:

- *Hide* – Oculta a janela do console do depurador e o painel das variáveis para maximizar o espaço de tela oferecido ao editor.
- *Pause* – Pausa imediatamente o aplicativo iPhone e entra no depurador.
- *Continue* – Executa o aplicativo até que outro breakpoint seja atingido.
- *Step Over* – Executa a próxima linha de código e retorna para o depurador.

- *Step Into* – Executa a próxima linha de código e retorna para o depurador. Se a linha chamar algum método, eles também poderão ser analisados passo a passo.
- *Step Out* – Continua a execução do código, até que o método atual retorne.

Seu breakpoint fez com que o depurador pausasse a execução no início de um sorteio simulado de cara ou coroa. Se você visualizar o painel de variáveis, ou se passar seu mouse sobre o argumento `userCalledHeads`, poderá determinar se o usuário escolheu cara (`YES`) ou coroa (`NO`).

A primeira linha do método `simulateCoinToss:` simula o lançamento de uma moeda (selecionando um número aleatório, 0 ou 1). Nesse ponto, o depurador está parado nessa linha (indicado pelo marcador vermelho na margem) e suas instruções ainda não foram executadas.

Para solicitar que o depurador execute uma única linha do código-fonte e depois retorne para o depurador, você pode clicar no botão **Step Over** e avançar à próxima linha do código-fonte. Isso fará com que o sorteio seja simulado, e o marcador vermelho deverá descer para a próxima linha de código-fonte. Nesse estágio, você pode determinar o resultado do sorteio passando seu mouse sobre o nome da variável `coinLandedOnHeads`; uma vez mais, `YES` significa cara, e `NO`, coroa.

Utilizando esse recurso mais algumas vezes, você pode percorrer as duas instruções `if`, atualizando o resultado e o estado dos `UILabel`s na interface do usuário. Ainda assim, diferentemente do que poderíamos esperar, se você verificar o dispositivo iPhone nesse momento, verá que os rótulos não foram atualizados. Isso ocorre pela forma como operam os componentes Cocoa Touch: a tela será atualizada apenas quando você liberar o depurador e permitir que esse método retorne ao sistema operacional.

Para permitir que o iPhone atualize a interface do usuário, e para visualizar as animações que indicam um novo resultado, você pode clicar em **Continue** (ou pressionar Cmd-Option-P) e permitir que o aplicativo continue sua execução, até que atinja outro breakpoint, ou até que você o pause novamente. Olhando mais uma vez seu iPhone, você poderá ver que os resultados do sorteio aparecem finalmente na tela.

1.7 Sumário

Parabéns, você desenvolveu seu primeiro aplicativo iPhone! Mostre o resultado para os seus amigos e a sua família. Talvez ele não venha a ser o mais novo sucesso da App Store, mas na criação desse aplicativo você aprendeu a dominar muitos dos recursos importantes do IDE Xcode, estando no caminho certo para desenvolver aplicativos de sucesso.

Ainda que Objective-C seja uma linguagem poderosa com muitos recursos, você verá que utilizar ferramentas visuais como o Xcode pode produzir grandes melhorias de produtividade, especialmente durante a prototipagem inicial de seu aplicativo. Desvincular a lógica de um aplicativo da forma como ela é apresentada ao usuário é um mecanismo poderoso que não deve ser subestimado. É provável que a primeira interface de usuário projetada por você não seja perfeita. Ser capaz de alterá-la, sem ter de modificar nenhuma linha de código, é uma vantagem poderosa.

Da mesma forma, você pode confiar no framework Cocoa Touch para trabalhar na implementação de muitos recursos de seu jogo. Por exemplo, as animações foram implementadas de modo declarativo: você especificou pontos de início e fim para as operações de rotação e esmaecimento e deixou que o framework trabalhasse para redesenhar a tela e efetuar a animação, acelerando ou desacelerando o efeito na medida em que esse se completava.

Como você continuará vendo, os frameworks Cocoa Touch são muito poderosos. Se você tiver de escrever grandes quantidades de código para recursos específicos, é provável que não esteja aproveitando tudo que o Cocoa pode lhe oferecer.

No capítulo 2, vamos nos aprofundar em tipos de dados, variáveis e constantes. Também seremos apresentados ao aplicativo gerenciador de aluguéis, que veremos no restante deste livro.